UNLIMITED
UNCLASSIFIED

ILLIMITÉ
NON CLASSIFIÉ

Technical Report

Rapport technique

1984/07

TR-SYS-005
NRC No. 23528

ALGORITHMS FOR SOLVING CRANE SCHEDULING PROBLEMS

Jiesheng Wang

Division of
Mechanical Engineering

Division de
génie mécanique

National Research Council Canada
Conseil national de recherches Canada

Canada

88-04469/

DIVISION OF MECHANICAL ENGINEERING PUBLICATIONS

DM	**(Division of Mechanical Engineering Report)** Scientific and technical information considered important, complete and a lasting contribution to existing knowledge.
TR	**(Technical Report)** Information less broad in scope but a substantial contribution to existing knowledge.
CTR	**(Controlled/Classified Technical Report)** A Technical Report with controlled distribution for national security, proprietary or other reasons.
LM	**(Laboratory Memorandum)** Preliminary or exploratory information with controlled distribution.
CAT	**(Calibration Analysis and Test Report)** Information on minor laboratory projects or services.

PUBLICATIONS DE LA DIVISION DE GÉNIE MÉCANIQUE

DM	**(Rapport de la Division de génie mécanique)** Informations scientifiques et techniques jugées importantes, complètes et susceptibles de contribuer de façon durable à l'avancement des connaissances courantes.
TR	**(Rapport technique)** Informations de moindre importance, mais pouvant contribuer substantiellement à l'avancement des connaissances actuelles.
CTR	**(Rapport technique à diffusion contrôlé/classifié)** Rapport technique à diffusion contrôlée pour des raisons de sécurité nationale, de propriété intellectuelle et autres.
LM	**(Mémoire de laboratoire)** Informations préliminaires ou de nature exploratoire à diffusion contrôlée.
CAT	**(Rapport d'étalonnage d'analyse et d'essai)** Informations sur de petits projets ou des services de laboratoire.

UNLIMITED
UNCLASSIFIED

ILLIMITÉ
NON CLASSIFIÉ

ALGORITHMS FOR SOLVING CRANE SCHEDULING PROBLEMS

ALGORITHMES POUR LA RÉSOLUTION DES PROBLÉMES DE LA PROGRAMMATION DES PONTS ROULANTS

Jiesheng Wang
Chinsese Academy of Sciences
Beijing, China

Technical Report

1984/07

Rapport technique

TR-SYS-005
NRC NO. 23528

P. Hamill, Head/Chef
Systems Laboratory/
laboratoire des systèmes

E.H. Dudgeon
Director/
Directeur

ACKNOWLEDGEMENT

I would like to express my gratitude to Leo Nenonen who has directed and assisted me in this project. I am happy to acknowledge the help received from Izhar Mufti, who has also contributed the OVERVIEW pp(iv) to (vii). Their suggestions, insights and critical comments greatly inspired me and led to numerous improvements in the design of algorithms. I would also like to express my appreciation to Albert Chan for many discussions.

ABSTRACT

A mathematical description is presented for formulating crane scheduling problems. Various versions of the problem are introduced. The general crane scheduling problem and some of its special cases belong to a class of problem for which no efficient algorithms are available for their solution. In this report, efficient approximate methods are described which can find good schedules not only for crane problems with equal job processing times but also for the general crane problem.

RÉSUMÉ

On présente une description mathématique pour la formulation des problèmes de la programmation des ponts roulants. Diverses versions du problème sont introduites. Le problème généralisé de la programmation des ponts roulants et certains de ses cas particuliers appartiennent à une classe de problèmes pour lesquels il n'existe pas d'algorithme de résolution efficace. Nous donnons dans le présent rapport des méthodes d'approximation efficaces permettant d'établir de bonnes programmations non seulement pour les problèmes à temps d'utilisation égaux mais aussi pour le problème généralisé.

AN OVERVIEW

by I.H. Mufti

As in the case of many other problems in the field of scheduling theory, the general crane scheduling problem belongs to a class of problems, called NP-complete problems. According to their degrees of difficulty, the computational problems can be classified into three main classes - (1) undecidable problems, (2) intractable problems and (3) NP problems. The problems belonging to the first category are those problems for which no algorithm can be written. The intractable problems, on the other hand, are solvable problems and do have algorithms but only inefficient ones running in exponential time. These are provably hard problems and no polynomial-time algorithm can possibly be developed to solve them. An algorithm is said to be a polynomial-time algorithm if its running time is bounded by a polynomial function p(n) of the input size n. Such algorithms are considered good or efficient, the linear-time $(p(n)=cn)$ being the most sought after.

We now consider the third category of problems, namely NP problems. Here NP stands for nondeterministic polynomial. The problem in this class can be solved in polynomial time if we can guess correctly the computational path that we should follow. This class includes as subclasses two important classes of problems, called P problems and NP-complete problems. The class P problems are tractable problems and have efficient algorithms, that is they can be solved using polynomial-time algorithms. However, NP-complete problems have exponential algorithms but we cannot prove that they cannot have polynomial-time algorithms. The importance of this class which includes such hard nuts as the travelling salesman problem, Hamilton circuit problem, lies in the fact that either there exist polynomial-time solutions for all of these problems or none of them has a polynomial-time solution. At this point in time, it is not known which of these two assertions is true.

To help understand why polynomial-time algorithms are considered good, let there be two algorithms A and B. The algorithm A needs 100 n^2 operations while B needs 2^n operations. If each operation takes one microsecond then algorithm A takes 0.25 second for $n=50$ and 1.05 seconds for $n=100$. On the other hand, algorithm B takes 1 second for $n=20$, 35 years for $n=50$ and 4.0×10^{14} centuries for $n=100$. It should be noted however, that for $n \leq 14$ algorithm B is faster than algorithm A.

As the general crane scheduling problem and some of its special cases are all NP-complete and cannot be solved exactly, one must look for approximate methods that could produce 'good' (perhaps not optimal) solutions in a reasonable amount of time. This report presents two methods which attempt to do just that. One of these methods is applicable to the general crane scheduling problem while the other is applicable to a special case with equal job processing times.

The crane scheduling problem is modelled using graph theory concepts and the methods presented in the report make extensive use of graph algorithms. As some of the readers may be unfamiliar with graph theory, the purpose of this review is to discuss some of the results in an informal way.

One of the fundamental concepts introducted in the report is that of the state of a job. Consider a job j consisting of k operations. The jobs are usually represented by k coordinates, $j=(\ell_1,\ell_2,\ldots,\ell_k)$ where each ℓ_i represents the job location of the ith operation of job j. As an example, j=(1,5) represents a two-operation job with job locations at 1 and 5. Suppose it takes 3 units of time to finish the first operation while it takes 2 units of time to finish the second operation. Then job j can also be written as $s_1=(1,1,1,5,5)$. Since crane idle time is allowed between the operations of the jobs we can also write job j as $s_2=(1,1,1,\phi,5,5)$ or $s_3=(1,1,1,\phi,\phi,5,5)$ where ϕ indicates a virtual operation (crane is idle). All these representations of the given job j=(1,5) are called the states of the job. The length of the state is the number of unit entries in that state. For example, the length of s_1 is 5 while that of s_3 is 7.

In this report, two methods are presented. the first method, called CCH (Chordal graph - Clique cover - Hamiltonian path) is applicable only to jobs with equal processing times and generates schedules which do not allow any crane idle time between the operations of the job. The number of operations per job and the operation processing times may be different for different jobs.

Suppose we have m cranes and job set $J = \{j_1,j_2,\ldots,j_n\}$ of n jobs. Let p denote the jobs processing time. Then, with the single track constraint removed (i.e., passing of the cranes allowed), since all jobs can be performed simultaneously we arbitrarily partition the jobs into $\lceil \frac{n}{m} \rceil$ batches, yielding the makespan (jobs completion time) $M^* = \lceil \frac{n}{m} \rceil p$ where $\lceil x \rceil$ denotes the least integer greater than or equal to x. However, in the presence of a single track constraint, where it may not be possible to perform all jobs simultaneously there arises the question of determining a minimum number of subsets of jobs such that the jobs belonging to each subset can be performed simultaneously. As the crane scheduling problem is modelled in terms of graphs, each subset, in the language of graph theory, is called a clique while the minimum number of subsets is called a minimum clique cover. The number of cliques in a minimum clique cover is known as the clique cover number. Hamilton path is needed to assign cranes to jobs in each subset.

Example: As an example, consider the case of two-operation jobs $J=\{(1,6),(2,4),(5,3),(9,10),(7,13)\}$. Let each operation take one unit of time so that the job processing time p=2 time units. By comparing the jobs processing locations, it is easy to check that jobs j_1,j_4 and jobs j_2, j_5 can be performed simultaneously. Other combinations of jobs are also possible. By determining all such combinations we find that the minimum number of subsets of jobs such that the jobs belonging to each set can be performed simultaneously is three. That is, the clique cover number is three. One such collection of subsets is $\{j_1 j_4\}$, $\{j_2, j_5\}$ and $\{j_3\}$.

The clique cover number can also be obtained in another way by determining the maximum number of jobs such that no two of them can be performed simultaneously. By comparing the job locations, it is easy to check that this number is three since no pairs of jobs from the set $\{j_1, j_2, j_3\}$ can be performed simultaneously. This maximum number, denoted by α, is known as the independence number.

The significance of the independence number α is that if for given n and m, $\alpha \geq \left\lceil \frac{n}{m} \right\rceil$, then the lowest achievable makespan is αp and increasing the number of cranes would not affect this number. However, if $\alpha \left\lceil \frac{n}{m} \right\rceil$ then the lowest makespan is $\left\lceil \frac{n}{m} \right\rceil$ which may be improved by increasing the number of cranes, since $\left\lceil \frac{n}{m_2} \right\rceil \leq \left\lceil \frac{n}{m_1} \right\rceil$, $m_2 > m_1$. Schedules with a makespan of 6 for m=2 and m=3 are shown in Fig. 1. Notice that for m=2, $\alpha=3=\left\lceil \frac{5}{2} \right\rceil$ and increasing the number of cranes from 2 to 3 does not lower the makespan, and crane 3 remains idle. The schedule with 2 cranes is optimal since its makespan $M=M^*=6$.

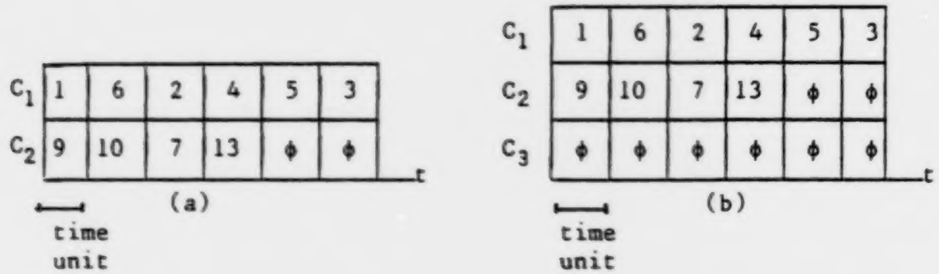
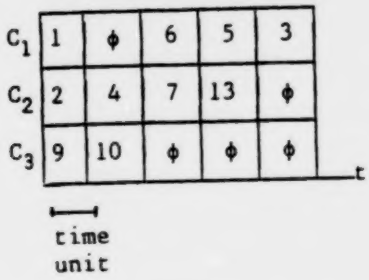


Fig. 1 Gantt chart (a) m=2, (b) m=3, makespan M=6

Note that for m=3, the makespan can be improved if we abandon the CCH method which restricts the space of feasible schedules by not allowing idle time between the operations of jobs. A schedule with makespan of 5 for m=3 is shown in Fig. 2.



C_1	1	ϕ	6	5	3
C_2	2	4	7	13	ϕ
C_3	9	10	ϕ	ϕ	ϕ

Fig. 2 Gantt chart, m=3, M=5

For simple problems such as the one illustrated, it is easy to determine the maximum independent sets and minimum clique covers. However, for the general job compatibility graph where vertices represent the jobs and edges between the vertices are drawn if and only if two jobs can be performed simultaneously there are no known efficient algorithms except for the case when the job compatibility graph is a chordal graph. To make use of the efficient algorithms available for chordal graphs, the author converts the job compatibility graph (if not already chordal) to a chordal graph by adding a certain number of edges and obtains a minimum clique cover for this new graph. This is accomplished by using Algorithms 3.2, 3.3 and 3.5. Finally, to assign the jobs in each of these subsets to the various cranes, a Hamilton path with minimum number of added edges is generated by using Algorithm 3.6. Depending on the number of added edges in the Hamilton path, the jobs belonging to each of the subsets must be further subdivided so that they can be performed simultaneously.

The second method called the PCMI (Path-Clique-Matching-Iterative) method is applicable to general scheduling problems and generates schedules that allow crane idle time between consecutive operations of jobs. This method is more involved than the CCH since more than one state per job is allowed. For example, the job (1,6) can have three states of length 3 namely, $(\phi,1,6)$, $(1,\phi,6)$ and $(1,6,\phi)$.

The first step in the PCMI method is to generate all the states of a job of specified length ℓ. This is accomplished by constructing the state tree (Algorithm 4.3). The next step determines K_2 (or more) subsets of jobs (Algorithm 4.2), each subset containing K_1 jobs (Algorithm 4.1) so that the jobs belonging to the same subset can be performed simultaneously. The numbers ℓ, K_1 and K_2 are the parameters of this method. This should be compared with the CCH method where ℓ=p (jobs processing time) is fixed, K_2 is the minimum number of subsets covering all the jobs and K_1, the number of jobs in each subset, is not prespecified. In the illustrative example, K_2=3 and the number of jobs in the three subsets are 2,2 and 1.

Following the above steps a partial schedule can be constructed. The scheduling of the remaining jobs and improving on the schedule so obtained is carried out by the matching and iterative improvement procedures using Algorithms 4.5 and 4.6. As the method is only an approximate method, it is not guaranteed to give an optimal schedule.

CONTENTS

Page

ACKNOWLEDGEMENT (ii)

ABSTRACT (iii)

AN OVERVIEW (iv)

ILLUSTRATIONS (ix)

TABLES (xiii)

1.0 THE CRANE SCHEDULING PROBLEM 1

1.1 The Crane Scheduling Problem 1
1.2 The Crane System 2
1.3 Assumptions, Constraints and Performance Measures 3

2.0 SOME EXTENDED CRANE SCHEDULING PROBLEMS 6

2.1 Single Crane Scheduling Problem, Including Travelling Time 7
2.2 Precedence Constrained Crane Scheduling Problems 7
2.3 Crane Scheduling With Individual Deadlines 8
2.4 Weighted Completion Time Scheduling 8
2.5 Relationship Between Crane Scheduling and Other Scheduling Problems 8

3.0 THE CCH (CHORDAL GRAPH - CLIQUE COVER - HAMILTON PATH) METHOD FOR CRANE SCHEDULING WITH EQUAL JOB PROCESSING TIME 11

3.1 Introduction 11
3.2 Job Compatibility and Comparability Graph Model 11
3.3 Some Sufficient and/or Necessary Conditions and the Lower Bound of Makespan 13
3.4 Chordal Graph Approximate Model 19
3.5 Nonchordal Circuit Recognition and Elimination 25
3.6 Minimum Covering of the Approximate Model by Cliques 43
3.7 Directed Hamiltonian Path of a Tournament 50
3.8 The CCH Method 54

4.0 THE PCMI (Path-Clique-Matching-Iterative) Method for General Crane Scheduling Problem 54

4.1 State and State Compatibility 54
4.2 N-Matrix 60
4.3 The Algorithms for Finding T-Directed K-Paths and K-Cliques 67
4.4 State Tree 72

CONTENTS (Cont'd)

Page

4.5 Algorithms for Examining the Suitability of Jobs to Blanks 77
4.6 Construction of Exchangable Job Set Algorithm and Iterative Improvement Procedure 82
4.7 The PCMI Method 90
4.8 Computational Considerations and Data Structures 97

5.0 REFERENCES 100

DOCUMENTATION PAGE

ILLUSTRATIONS

Figure

1 The Geometric Representations of Jobs 3
2 Crane Interference and Idleness 4
3 Schedules and Makespans 5
4 Two Kinds of Directed Tree 8
5 The Schedule of a Special Job-Shop Scheduling Problem Obtained by Converting a Crane Scheduling Problem 10
6 The Schedule of the Crane Scheduling Problem 10
7 Unitary Jobs 12
8 Job Compatibility Graph and Digraph 13
9 The Directed Paths in a Job Compatibility Digraph 17
10 The Minimum Clique Covers and Maximum Independent Sets in a Job Compatibility Graph 17
11 Schedule S* With m=2 and Given Jobs 18
12 A Schedule S With m=3 and The Same Jobs 18
13 A Schedule S' With M=4 and The Same Jobs 18
14 A Compatibility Graph and It's Chordal Graph Approximation 19
15 A Comparability Graph and It's Transitive Orientation 21

CONTENTS (Cont'd)

Figure Page

16 A Chordal Graph Approximation of G in Fig. 15 is Also A Comparability Graph 21

17 A Comparability Graph G 21

18 A Chordal Graph Approximation of G in Fig. 17 is Also A Comparability Graph 22

19 A Chordal Graph Approximation of G in Fig. 17 is Not A Comparability Graph 22

20 A Comparability Graph Whose Two Possible Chordal Graph Approximations With The Minimum Added Edges Are Not Comparability Graph 22

21 Two Possible Chordal Graph Approximations of the Graph In Fig. 20 22

22 A Comparability Graph G Has Only One Implication Class, D' is the Corresponding Orientation 24

23 A Chordal Graph Approximation of G Whose Transitive Orientation $D' \supset D$ 25

24 A Graph and All Its Circuits of Length Strictly Greater Than 3 26

25 The Chordal Graph Approximations With 1,2,3 and 4 Added Edges Respectively 27

26 Two Examples of Vertices Which Cannot Belong To Any Nonchordal Circuits 28

27 An Example of Applying Lemma 3.2 29

28 Four Possible Cases of $P_1(K) \wedge P_2(K)$ 30

29 A Graph G and the Reduced Graph of G 32

30 The Evolution of Applying Algorithm 3.2 to G Shown in Fig. 29 and the Resulting Nonchordal Circuits 33-35

31 A Graph and Its Nonchordal Circuits 39

32 The Common Chord Sets of the Nonchordal Circuits 41

33 The Weights and Their Selection and Elimination 42

CONTENTS (Cont'd)

Figure Page

34 The Resulting Chordal Graph From Algorithm 3.3 42

35 A Graph With Six Simplicial Vertices 43

36 A Graph With the Minimum Clique Cover $C_1=v_1UV(v_1)$ and $C_2=v_2UV(v_2)$ 44

37 An Example That There Exists a Minimum Clique Cover Which Cannot Be Found By the Algorithm Applied to the Graph Shown in Fig. 36 45

38 A Graph and Its Chordal Graph Approximation G' With Two Added Edges 47

39 The Evolution of Applying Algorithm 3.4 to G' 49

40 Another Graph G' With 3 Added Edges 49

41 The Evolution of Algorithm 3.4 For G' Shown in Fig. 40 49

42 A Tournament 52

43 Finding Hamiltonian Path in a Tournament By Algorithm 3.6 53

44 The Various Structuring of Schedules 55

45 The Extended Jobs and Unitary Jobs 56

46 A Job and Its State Set 57

47 The Jobs and Their State Compatibility Graph and Digraph 58

48 An Example About the Relation $\overset{s}{\rightarrow}$ Between Two Job Sets 59

49 Two State Sets and State Compatibility Digraph 60

50 An Optimal Schedule For An Example 61

51 A Graph G for Illustrating Definition 4.9 and Algorithm 4.2 63

52 A Digraph D for Illustrating Definition 4.10 and Algorithm 4.1 65

53 An Illustration of Applying Algorithm 4.1 to State Compatibility Graph 68

CONTENTS (Cont'd)

Figure Page

54 Two Jobs With $n_j=3$, $p_j=6$ and Their State Sets With $\ell=p_j+1$ 73

55 A State Set Having the Same Structuring in Fig. 54 73

56 A State Set and The State Tree 75

57 A Schedule Having General Structuring 77

58 The Evolution of an Example for Applying Algorithm 4.4 81

59 Regard a Feasible Assignment As A Matching 82

60 A Schedule For Illustrating the Meanings of F,P,B,C,D 84

61 The Digraph D' Consisting of Unassigned Jobs and Exchangeable Jobs 85

62 An Example That There are Different Directed Paths Consisting of the Same Jobs 86

63 A Special Multi-Partite Digraph for Which Algorithm 4.1 is Available 87

64 An Illustration of the Iterative Improvement Procedure 89

65 The Classification of Schedule Structuring 91

66 The Comparison of Various Methods 92

67 A Case of Producing a Longer Idle Interval 93

68 An Example Making A Schedule Active 93

69 The Values of n_g, r, n_s 97

A.1 A Graph and Its An Induced Subgraph A-3

A.2 Vertex Removal A-4

A.3 The Matrix Representations of Graphs A-4

A.4 Adjacency Lists A-5

A.5 The Walk, Trail and Path of a Graph A-5

CONTENTS (Cont'd)

Figure		Page
A.6	The Directed Path	A-6
A.7	Hamiltonian Circuit and Hamiltonian Path	A-7
A.8	A Tournament	A-8
A.9	A Bipartite Graph and a Complete One	A-8,9
A.10	The Comparability Graph and Chordal Graph	A-10
A.11	A Graph With a Maximum Independent Set $\{b,d,f\}$	A-11
A.12	A Graph With the Minimum Clique Cover Number K=2	A-12
A.13	A Graph And Its Minimum Matching	A-12
A.14	A Graph With Chromatic Number X=3	A-13
B.1	Graph G and Its Implication Classes	B-1
B.2	G-Decomposition and Decomposition Scheme	B-2
B.3	A Graph Which Is Not a Comparability Graph	B-3
B.4	Some Graphs and Their Simplicial Vertices	B-4

TABLES

Table		
1	The One-Operation Jobs For Given Crane Scheduling Problem	9
2	The Jobs For An Example Illustrating Theorem 1.1	16
3	The Common Chord Set and Their Subsets	39
4	The Interference Subsets and Their Unions	40

SYMBOLS

Symbol	
α	The Criteria of Performance of a Block in a Schedule
$\alpha(G)$	The Independence Number of G
β	The Measure of Approximation of a Chordal Graph Model

CONTENTS (Cont'd)

Symbol	
Γ	A Binary Relation on Edge Set
δ	The Decomposition Scheme
$K(G)$	The Clique Cover Number of G
λ	The Perfect Elimination Scheme
$\lambda(i)$	The ith Simplicial Vertex in λ
μ	The Crane System
σ	A One-To-One Function $j \to Z_0^+$, $\sigma(j)$ is the Start Time of j
τ	The Time Unit
ϕ	The Idle Time (i.e. A Virtual Operation)
Φ	The Empty Set
$\chi(G)$	The Chromatic Number of G
a_j	(i) A r-bit Boolean Vector With a 1 in jth Position And 0 In All Other Positions (ii) The Lateness of j
A	The Arc Set of Directed Graph
$A(G)$	An Upper Triangular N-Matrix For Finding Cliques of G
$A(D)$	An N-Matrix For Finding T-Directed Paths of D
$A^{(K)}(G)$	K-Clique Matrix of G
$A^{(K)}(D)$	K-T-Directed Paths Matrix of D
b	A Blank In a Schedule
$b(J(j))$	The Cells in a Schedule for Jobset J(j)
B	The Set of Blanks
B_i	(i) The ith Blank of S (ii) An Implication Class of a Graph
B_{ij}	An Implication Class of A Graph
B_p	The Block Produced By T-Directed Path p

Symbol	
$B(G)$	A Lower Triangular N-Matrix for Finding Cliques of G
$B^{-}(D)$	An N-Matrix For Finding T-Directed Paths of D
$B(n,r)$	The Set of the Distinct Blanks
c_j	The Completion Time of Job j
C	(i) The Crane Set (ii) A circuit (iii) A Clique Cover
C_j	(i) The jth Crane (ii) A circuit (iii) A Clique
$C(j)$	The Exchangeable Set of Job j
$C(k)$	The K-Circuit Set
CCH	The Chordal Graph - Clique Cover - Hamiltonian Path Method
d_j	(i) The Deadline of Job j (ii) A Common Chord (iii) The Degree of Vertex v_j
$d_j^{-}, d^{-}(v_j)$	The Degree of Vertex v_j
$d_j, d(v_j)$	The In-Degree of Vertex v_j
D	A Directed Graph
D_i	The Common Chord Set In Circuit C_j
D_J	The Subgraph of D Induced by J
$D(j)$	The Set of Cells For J(j)
$D=(V,A)$, $D(V,A)$	A Directed Graph With Vertex Set V and Arc Set A
E	The Edge Set of A Graph
E_i	A Partition Element of E, i.e. $E=\sum_{i=1}^{k} E_i$
F	The Set of Fair Blocks in a Schedule
$g(d)$	The Number of Saved Chords

CONTENTS (Cont'd)

Symbol	
$G=(V,E)$, $G(V,E)$	A Graph With Vertex Set V and Edge Set E
$G=(X,Y)$	The Job-Blank Suitability Bipartite Graph
I_i	(i) The Total Idle Time of Crane C_i (ii) Moving Interval Of Operation O_i in a State
$I_i(d)$	The Interference Set of d in D_i
j	A Job
J	The Job Set
J_b	The Set of Suitable Jobs for b
$J(j)$	The Set of Exchangeable Jobs of j
K1	A Parameter of PCMI Method, i.e. The Number of Vertices of the T-Directed Paths
K2	A Parameter of PCMI Method, i.e. The Clique Size
K-Circuit	A Circuit with K Vertices
K-Path	A Path With K Vertices
K-Clique	A Clique With K Vertices
ℓ	The State Length
ℓ_i	(i) The ith Location on the Aisle (ii) The Processing Location of the ith Operation of a Given Job
ℓ_{ij}	The Processing Location of the ith Operation of j
L	The Location Set
L_j	The Processing Location Sequence of j
$L(P_i(K),t),L(i,t),L(P_i,t)$	The Label of Path $P_i(K)$ at Time t
$L(v_{ij})$	The Label of Vertex v_{ij} in a State Tree
m	The Number of Cranes

CONTENTS (Cont'd)

Symbol	
$m(d)$	The Multiplicity of Common Chord d
M	The Makespan of a Schedule
$\overline{M}$	The Minimum Makespan Achieved Probably
M^*	The Makespan of an Optimal Schedule
n	(i) The Number of Jobs (ii) A Structuring Parameter of State Tree
n_j	The Number of Operations of j
N	The Number of Locations
N_p	The Number of State-Disjoint Paths
Ns	The Total Number of States for all Jobs
$N(t)$	The Number of Components of $L(P_i(K),t)$
NP	The Class of Nondeterministic Polynomial-Time Problems
o_i	The ith Operation of a Job
$O(f(n))$	Complexity on the Order of f(n) Where n is the Number of the Input
p	The Processing Time of a Given Job, i.e. The Job Length
p_j	The Processing Time of j
p_{ij}	The Processing Time of Operation o_i of j
P	The Set of Poor Block in a Schedule
P_j	The Processing Time Sequence of j
PCMI	The Path-Clique-Matching-Iterative Method
Q	The Intersection of P and $Q(n,r)$
$Q(n,r)$	The Job-Blank Set
r	A Structuring Parameter of State Tree

CONTENTS (Cont'd)

Symbol	
r_j	The Ready Time of j
R	The Subset of Distinct Member of R'
R'	The Set of Structuring Parameters of State Tree
s	A State
s_f	The Number of Fair Blocks
S	(i) The State Set (ii) A Schedule
S_{jb}	The Set of States of j Being Suitable for b
S(j)	The Set of States Belonging to j
t_i	The Start Time of Operation o_i in a State
t_j	The Tardiness of j
$t(\ell_i,\ell_j)$	The Moving Time From Location ℓ_i to ℓ_j
T	The Set of State Trees
T(n,r)	The State Tree With Structuring Parameters n and r
T-Directed	Transitive Directed
U(v), U_v	The Set of Neighbours of V
$U^-(v),U^-_v$	The Set of Vertices u's, $(u,v)\epsilon A$
V	The Vertex Set of a Graph
V_p	The Set of Vertices of Path p
V(B)	The Vertex Set Consisting of the Jobs (Vertices) in B
w_j	The Weight of job j
w(d)	The weight of Chord d
X	(i) A Partition Element of a Bipartite Partition of G(X,Y) (ii) The Lower Bound Vector of b

CONTENTS (Cont'd)

Symbol	
Y	(i) A Paritition Element of a Bipartite Partition of G(X,Y) (iii) The Upper Bound Vector of b
Z^+	The Set of All Positive Integers
Z_o^+	The Set of All Non-Negative Integers
$\forall x$	For All x
$A \subseteq X$	A is a Subset of X
$B \subset X$	B is a Proper Subset of X
$a \in A$	a is a Member of X
$\lvert X \rvert$	The cardinality of a Set X
$A \cap B$	The Intersection of A and B
$A \cup B$	The Union of A and B
$P_i \wedge P_j$	The Overlap of Paths P_i and P_j
$\{a_1,a_2,\ldots,a_k\}$	The Set Consists of $a_1,a_2,\ldots,a_k$
$(a_1,a_2,\ldots,a_k)$	A Sequence Consisting of $a_1,a_2,\ldots,a_k$
(S,m)	A 2-Tuple Consists of S and m
(u,v)	A Arc of a Digraph
uv	A Edge of a Graph
$\lceil x \rceil$	The Ceiling Function, i.e. the Least Integer Greater Than and Equal to x
$G-\{v\}$	Vertex Removal
$j_1 \prec j_2$	j_2 Does Not Begin Execution Before j_1 Finished
$\underset{\sim}{j}, \underset{\rightarrow}{j}$	Two compatibility Relations on Job Set J
$\underset{\sim}{s}, \underset{\rightarrow}{s}$	Two compatibility Relations on State Set S

ALGORITHMS FOR SOLVING CRANE SCHEDULING PROBLEMS

1.0 THE CRANE SCHEDULING PROBLEM

1.1 Introduction

Efficient material handling systems have great significance for the productivity of large industrial plants, such as steel mills and copper smelters. In such plants, the molten materials are generally transferred in ladles by overhead cranes. There is only one track shared by all cranes on each crane aisle. Each crane can move to any location in the aisle, but no passing is possible. If a crane was assigned to do a job and a working crane is in the way, a crane interference will occur and the material transfer will be delayed. These delays can seriously reduce the productivity of the plant.

In actual practice some conditions are dynamic, e.g., personnel, equipment allocations or process rates etc. change. A computer simulation model has been developed for copper smelters [Ref. 1] which can be used to test the outcomes of various operating policies. As conditions change in the plant, the model description is updated. Alternate equipment arrangements can be evaluated before they are implemented in the smelter. The model can answer "what if" questions without disrupting production.

Crane scheduling problems are scheduling problems with special constraints. The general crane scheduling problem and some restricted crane problems have been shown to be NP-complete [Ref. 2]. The crane scheduling problem with one operation per job has been investigated [Ref. 3]. A "batching procedure" for two-operation jobs with equal operation processing times and "meshing procedure" for two-operation jobs with 2 cranes and equal operation processing times has been developed [Ref. 4]. In this report, the following work will be described:

(1) introduction of new crane scheduling problems;
(2) establishment of graph theory models;
(3) presentation of the "Chordal Graph - Clique Cover - Hamiltonian Path" (CCH) method and some sufficient or necessary and sufficient conditions for the crane scheduling problem with equal job processing times;
(4) description of the N-matrix concept and the "T-directed Path - Maximum Clique - Maximum Matching - Iterative Improvement" (PCMI) method for solving general crane scheduling problems.

The PCMI method has been programmed in VAX FORTRAN language and has been applied to some test problems.

1.2 The Crane System

In most plants cranes share a common aisle and no passing is possible. That is if there are two cranes, one of them must always be the leftmost crane and the other must always be the rightmost crane.

Jobs form the input to a crane system and each job is made up of one or more operations. All operations are performed at discrete locations along the crane aisle and one operation is performed at only one location. For any job, the same crane must perform all of its operations, and the operations must be processed in a given order.

There are many parameters involved in crane scheduling problems. We introduce the following symbols to represent these parameters.

m	the number of cranes
n	the number of jobs
N	the number of locations
n_j	the number of operations for job j
r_j	the ready time for job j
$C = (C_1, C_2, \dots C_m)$	the crane sequence
$J = \{j_1, j_2, \dots j_n\}$	the job set
$L = (1, 2, \dots N)$	the location sequence
$L_j = (\ell_{j1}, \ell_{j2}, \dots \ell_{j,nj})$	the processing location sequence for job $j \in J$, where $\ell_{ji} \in L$ is the processing location of ith operation of the job $(i=1,2,\dots n_j)$
$P_j = (p_{j1}, p_{j2}, \dots p_{j,nj})$	the processing time sequence for job $j \in J$ where p_{ji}, a positive integer, is the number of time units required to perform ith operation of the job $(i=1,2,\dots n_j)$
$p_j = \sum_{i=1}^{n_j} p_{ji}$	is the total processing time of job j, that is, the length of the job.

A crane system can be represented by the six-tuple

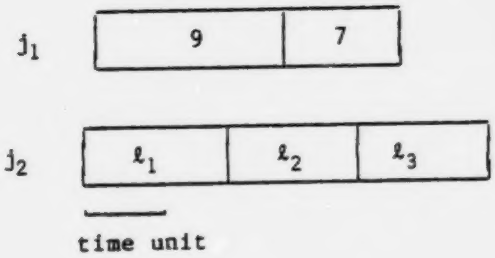
$$\mu = \langle m, N, (n, n_j), r_j, \ell_{ji}, p_{ji} \rangle$$

which means that there are m cranes, N valid aisle locations, n jobs to be done with each job $j = 1,n$ having n_j operations and being ready at time r_j. Furthermore, each job j features a set of locations ℓ_{ji} for each operation i in the job, $i = 1, n_j$ and an associated processing time p_{ji} for each operation i.

Usually, we use the processing location sequence L_j to represent job j, for example

$j_1 = (3,5)$
$j_2 = (\ell_1, \ell_2)$

Each job can also be described geometrically. Some examples are given in Fig. 1.



$L_1 = (9,7)$ $L_2 = (\ell_1, \ell_2, \ell_3)$
$P_1 = (3,2)$ $P_2 = (2,2,2)$
$p_1 = 5$ $p_2 = 6$

Fig. 1

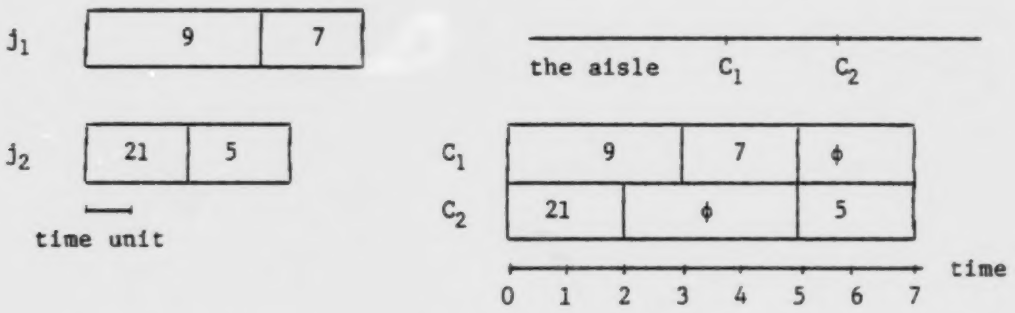
1.3 Assumptions, Constraints and Performance Measures

Any scheduling problem involves some specific assumptions and constraints and the crane scheduling problem is no exception. The following assumption will be made in defining the basic crane scheduling problem.

(1) Any crane is suitable for any operation and uses the same processing time.
(2) No pre-emption for each job. Each job, once started using a crane, must be completed by the same crane before the crane may start another job.
(3) Continuity of processing for each operation. Once started, an operation cannot be stopped before it is finished.
(4) Zero ready time (i.e. all jobs are ready to be done now).
(5) Zero travelling time.
(6) Equal job lengths. $p_j = p$ (constant) for all $j \in J$.

In addition, the major constraint featured by the basic crane scheduling problem is that there can be no passing of cranes. We number all cranes in sequence from left to right, that is, the leftmost crane is C_1, the first on the right of C_1 is C_2, the second is C_3, and so on. Then, the constraint means that crane C_i is on the left of C_{i+1} and there can be no crane between them at any time. Assumptions (1) to (3) and the no passing constraint must be met for all crane scheduling problems we consider. In certain cases, assumptions (4)-(6) need not be met.

The no passing constraint is the major difference between the crane scheduling problem and the general scheduling problem. This constraint can cause crane interference and idleness as illustrated in Fig. 2. If cranes C_1, C_2 are committed to process jobs j_1, j_2 respectively and the two jobs are started simultaneously at time 0, then, crane interference occurs at time 2.



(a) two jobs

(b) a Gantt Chart
ϕ represents idleness

Fig. 2

Having finished the first operation of j_2, C_2 should move from location 21 to 5 for processing the second operation. To do so, C_2 would have to cross C_1 (while C_1 is at location 9 to process j_1), but it is not allowed. The crane interference causes C_2 to be idle for 3 time units (marked by ϕ).

Due to the no passing constraint the set of feasible solutions of crane scheduling problems is very complex making structuring of the set very difficult.

Solving crane scheduling problems involves scheduling the processing of jobs by cranes so that some measure of performance achieves its optimal value. However, it is not easy to state our objectives in scheduling because they are numerous, complex and often conflicting. Mellor lists 27 distinct scheduling goals, albeit in a slightly broader context [5]. No matter what objective is selected, we will obtain an optimization problem. In general, the problem will be extremely difficult mathematically, even with the simplest of objectives. We shall not try to discuss all of the objectives but will only discuss a few of the criteria.

Before we can define performance measures in precise mathematical terms, we need some more definitions and notations:

D Overall deadline, i.e. the promised delivery date of j, $\forall j \in J$. It is the time by which we would like to have completed all jobs, ideally.

d_j	Deadline of job j, i.e. the promised delivery date of $j \in J$.
σ^j	For a given schedule, σ is a one-to-one function σ: $J \rightarrow Z_o^+$, where J is the job set, Z_o^+ is the non-negative integer set $Z_o^+ = \{0,1,2,...\}$. $\sigma(j)$ means that $j \in J$ will be started at time $\sigma(j)$ according to the schedule (see Fig. 3, $\sigma(j_1) = 1$, $\sigma(j_2) = 5$, $\sigma(j_3) = 0$, $\sigma(j_4) = 5$, for schedule 1).
w_j	Weight of job j. $w_j \in Z^+$, Z^+ is the positive integer set.
e_j	Delay of job j. That is, e_j is the sum of the waiting times between successive operations of the job j due to interference.
c_j	Completion time of job j, i.e. $c_j = \sigma(j) + e_j + p_j$ where p_j is the total processing time of j. Note that for PCMI method (see section 4), c_j still contains the idle time between two operations.
I_i	Total idle time of crane C_i. There are two types of idleness: (1) Waiting between two successive operations; (2) the idleness between the time at which the crane finished the last job assigned to it and the time at which all jobs are finished. I_i is the sum of these idle periods. Obviously, I_i depends on the schedule we are considering.

Now, we will define some measures of performance.

1. Makespan. If the cost of a schedule depends on how long the crane system is occupied by the entire set of jobs, we can minimize c_{max}, where $c_{max} = \max c_j$, $j \in J$. c_{max} will be called the makespan M of the schedule, sometimes it is called the total completion time. As an example, see Fig. 3.

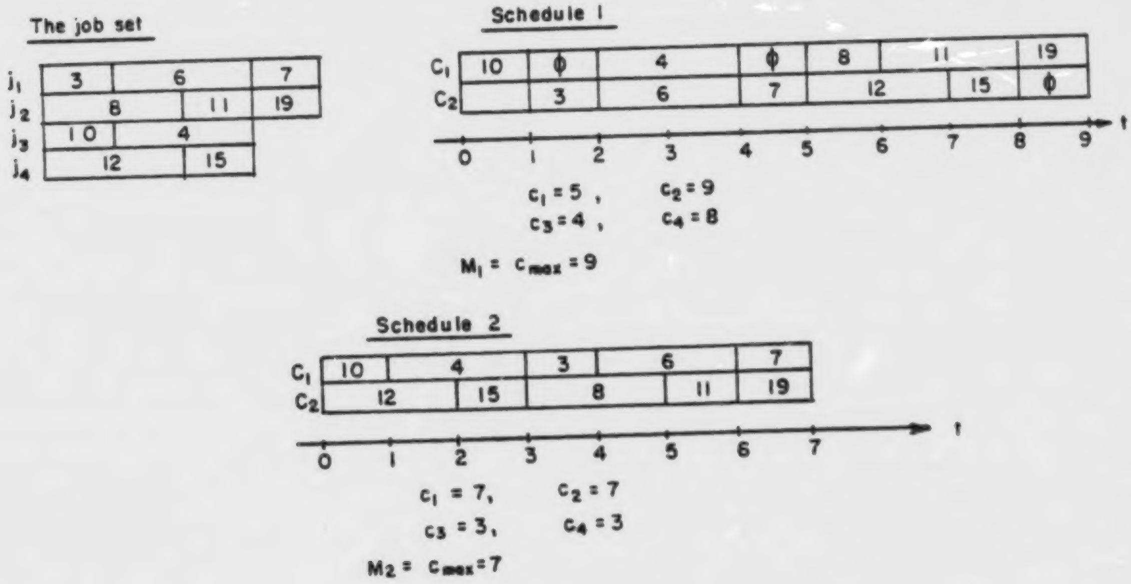

The job set
j_1
j_2
j_3
j_4
3
6
7
8
11
19
10
4
12
15
Schedule 1
C_1
C_2
10
ϕ
4
ϕ
8
11
19
3
6
7
12
15
ϕ
0
1
2
3
4
5
6
7
8
9
t
$c_1 = 5$, $c_2 = 9$
$c_3 = 4$, $c_4 = 8$
$M_1 = c_{max} = 9$
Schedule 2
C_1
C_2
10
4
3
6
7
12
15
8
11
19
0
1
2
3
4
5
6
7
t
$c_1 = 7$, $c_2 = 7$
$c_3 = 3$, $c_4 = 3$
$M_2 = c_{max} = 7$

Fig. 3

There are two schedules in Fig. 3. Obviously schedule 2 is an optimal schedule with minimum makespan $M^* = 7$.

It is appropriate to mention here that a schedule has minimum makespan if it has minimum total idle time $I = \sum_{i=1}^{m'} I_i$, if for the associated schedules, exactly $m' \leq m$ cranes are used.

2. Lateness and Tardiness. We should first define lateness and tardiness. The lateness a_j of $j \varepsilon J$ is simply the difference between its completion time c_j and its completion deadline d_j: $a_j = c_j - d_j$. Note that when a job is completed early, i.e. when it is completed before its deadline, a_j is negative. It is often more useful to consider only non-negative values when a job is tardy, i.e. when it is completed after its deadline. Hence, we can define tardiness $t_j = \max\{a_j, 0\}$.

Assume that a_{max} and t_{max} are the maximum lateness and maximum tardiness, respectively. If there is a positive reward for completing a job early, we can take minimum a_{max} as our performance measure. When early job completions bring no reward; but penalties are incurred for late jobs, minimizing t_{max} is appropriate.

3. Weighted Completion Time, Weighted Lateness and Weighted Tardiness. In cases, where we wish to assign a weight w_j for job j, we can use the following weighted completion time, weighted lateness and weighted tardiness instead of c_j, a_j and t_j, respectively.

weighted completion time:

$$c_j w_j = (\sigma(j)+e_j+p_j) \cdot w_j \tag{1}$$

weighted lateness:

$$a_j w_j = (\sigma(j)+e_j+p_j-d_j) \cdot w_j \tag{2}$$

weighted tardiness:

$$t_j w_j = \max\{(\sigma(j)+e_j+p_j-d_j), 0\} \cdot w_j \tag{3}$$

2.0 SOME EXTENDED CRANE SCHEDULING PROBLEMS

For the general scheduling problems, the single machine processing problem is studied in great detail. The case in which there is only one machine involved is the simplest of scheduling problem. It is interesting that single machine scheduling problems arise in practice more often that one might expect, e.g. the processing of jobs through a small non-time-sharing computer is a single machine processing problem, or, there are job shops with more than one machine, but in which one machine acts as a "bottle-neck". Studying such simpler cases can give some information about complex problems and show us how difficult it is to solve a general problem.

2.1 Single Crane Scheduling Problem, Including Travelling Time

Define travelling time: $t(\ell_i,\ell_j)$ is the time needed for the crane to move from location ℓ_i to ℓ_j. The extended single crane scheduling problem is stated as follows: given the crane system $\mu = <1, N, (n,n_j), r_j, \ell_{ji}, p_{ji}>$ with non-zero ready times r_j, is there a schedule σ that meets the overall deadline D and minimizes the maximum completion time involving both processing time and moving time. That is, is there a schedule σ, such that for any two jobs j', j processed successively, where the completion time for job j is

$$c_j = \sigma(j) + e_j + p_j + \sum_{i=1}^{n_j-1} t(\ell_{j,i},\ell_{j,i+1}) + t(\ell_{j',n_{j'}},\ell_{j,1}) < D \qquad (4)$$

which makes $c_{max} = \max_{j \in J} c_j$ achieve the minimal value.

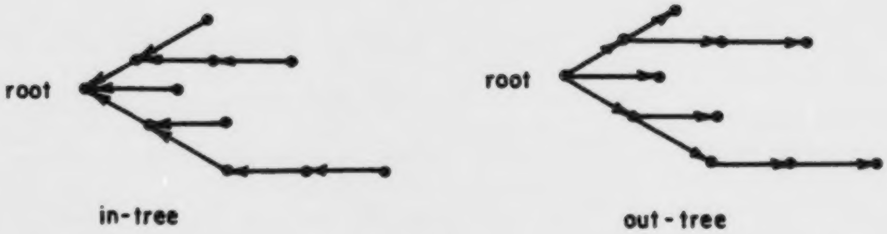
Note that $\sum_{i=1}^{n_j-} t(\ell_{j,i},\ell_{j,i+1})$ = constant for given job j and does not include any travelling time due to inteference. Note also that the $\sum_{i=1}^{n_j-1} t(\ell_{j,i},\ell_{j,i+1})$ represents the total travelling time required between the locations of the n_j operations comprising job j and $t(\ell_{j',n_j'},\ell_{j,1})$ represents the travelling time from the location $\ell_{j',n_j'}$ of the final operation of the preceding job j' to ℓ_{j1}, the location of the first operation for job j.

2.2 Precedence Constrained Crane Scheduling Problems

In scheduling theory, a significant subject is the scheduling problem with precedence constraints for which some important results have been obtained. The most fundamental of them is the Hu algorithm for scheduling equal length tasks on m identical processors subject to treelike precedence constraints, in order to minimize the makespan [6]. The generalization of Hu's algorithm which allows arbitrary acyclic precedence constraints has been shown to be NP-complete [7,8].

Because precedence constraints are encountered in most industrial plants, we should consider the corresponding crane scheduling problems.

Precedence Constraint: the constraint is defined as a partial order on job set J. For $j, j' \in J$, $j < j'$ implies $\sigma(j') > c_j = \sigma(j) + e_j + p_j$. The common partial orders are "treelike" orders. There are two basic types of such partial orders. The first, called an in-tree partial order, is derived from a rooted directed tree in which every vertex lies on a path directed to the root. The second, called an out-tree partial order, is given by a rooted directed tree in which every vertex lies on a directed path from the root. Figure 4 gives an example for both types of directed tree.



in-tree

out-tree

Fig. 4

Precedence Constrained Crane Scheduling Problems: Given assumptions (1)-(3) and crane system $\mu = \langle m,N,(n,n_j),0,\ell_{ji},p_{ji}\rangle$, is there a schedule σ which meets the no passing constraint, the overall deadline D and obeys given precedence constraint? Note that this formulation does not consider travelling times and does not allow delays between operations within a job.

2.3 Crane Scheduling with Individual Deadlines

Sometimes both precedence constraints and individual deadlines are simultaneously imposed on a crane scheduling problem.

Individual deadlines. Each job j has a deadline d_j, i.e. for any feasible schedule σ, $c_j < d_j$, $\forall j \in J$, $(d_j \in Z^+)$. Note again that crane travel times are not considered (i.e. assumed to be zero).

Individual deadline crane scheduling problems. Given a crane system $\mu = \langle m,N,(n,n_j),0,\ell_{ji},p_{ji}\rangle$, is there a schedule σ which meets assumptions (1)-(3) and the no passing constraint, individual job deadlines and obeys given precedence constraint?

2.4 Weighted Completion Time Scheduling

In the last section, we introduced the concept of a weight index w_j for job $j \in J$. Now, we can define the following optimization problem.

Scheduling to minimize weighted completion time. Given the crane system $\mu = \langle m,N,(n,n_j),0,\ell_{ji},p_{ji}\rangle$, is there a schedule σ that meets assumptions (1)-(3) and the no passing constraint and minimizes the weighted completion time, i.e. minimizes the sum $\sum_{j\in J} c_j \cdot w_j$? (Note again that travelling times are assumed to be zero). Or in a decision problem form: Given crane system μ and a positive integer K, is there a schedule σ that meets assumptions (1)-(3) and the no passing constraint such that the sum for all $j \in J$ of $c_j \cdot w$, is not more than K?

2.5 Relationship between crane scheduling and other scheduling problems

Finally, we wish to discuss the relation between crane scheduling and other scheduling problems. First, we consider the general job-shop scheduling problem. Suppose that we have n jobs $j_1,\ldots j_n$ to be processed through m machines, $M_1,M_2\ldots M_m$. Each job must pass through each machine once and only once and each job can be processed through the machines in arbitrary order. We wish to find a sequence by which the jobs pass through the machines which is optimal with respect to some criterion of performance.

We make the following restriction or change to the general job-shop scheduling problem:

(1) Each job passes through some of the machines in a particular order;
(2) The no. of machines working simultaneously is not more than a given positive integer $m' < m$.

In doing so, we will have a special type of machine scheduling problem.

The one-operation job and 2-operation job crane scheduling problems with equal operation processing times have been studied ([3],[4]). The two crane scheduling problems can be represented in the form of the special machine scheduling problems.

As an example, we convert the one-operation job crane scheduling problem into the special machine scheduling problem. This system is represented by $\mu = <m,N,(n,1),0,\ell_j,\tau>$ in which we have m cranes, N valid locations, n jobs each with 1 operation, all jobs are ready to be done now, jobs $j = 1,n$ are located at ℓ_j, and all have the same processing time τ. In terms of the special machine scheduling problem, we suppose that there are N machines, numbered 1,2,...N. Each job is processed by one specific machine ℓ_j. The integer m' of (2) above is equal to m. The conversion can be illustrated by Table 1 and Fig. 5,6.

j	ℓ_j
1	15
2	9
3	7
4	11
5	30
6	25
7	16
8	8
9	24

j	ℓ_j
10	28
11	5
12	27
13	13
14	18
15	10
16	22
17	19
18	20

$p_j = \tau$ (constant)
$m = 6$
$N = 30$
$n = 18$

Table 1

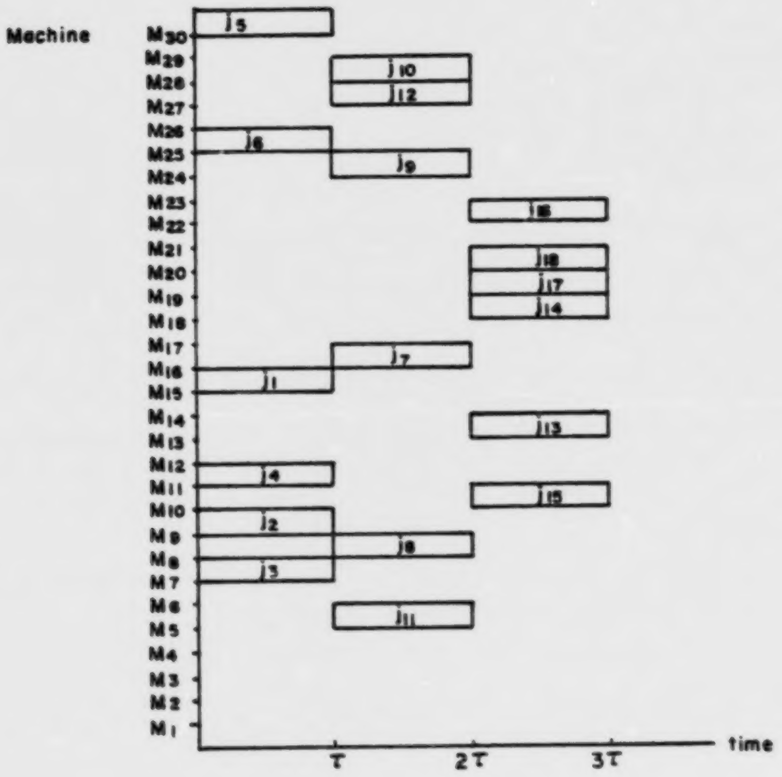


Fig. 5

Crane

C_1	$\ell_3 = 7$	$\ell_{11} = 5$	$\ell_{15} = 10$
C_2	$\ell_2 = 9$	$\ell_8 = 8$	$\ell_{13} = 13$
C_3	$\ell_4 = 11$	$\ell_7 = 16$	$\ell_{14} = 18$
C_4	$\ell_1 = 15$	$\ell_9 = 24$	$\ell_{17} = 19$
C_5	$\ell_6 = 25$	$\ell_{12} = 27$	$\ell_{18} = 20$
C_6	$\ell_5 = 30$	$\ell_{10} = 28$	$\ell_{16} = 22$
	τ	2τ	3τ time

Fig. 6

Table 1 lists the jobs and processing locations. Figure 5 shows a schedule for the resulting special machine scheduling problem and the corresponding schedule for the crane problem is shown in Fig. 6.

3.0 THE CCH (CHORDAL GRAPH - CLIQUE COVER - HAMILTONIAN PATH) METHOD FOR CRANE SCHEDULING WITH EQUAL JOB PROCESSING TIMES

3.1 Introduction

One of the simplest crane scheduling problems is that with one-operation jobs. It has been shown that the one-operation crane scheduling problem with arbitrary processing times is NP-complete, and an algorithm is given which constructs a feasible solution in linear time, but does not guarantee an optimal solution. For the case with equal processing times, an optimal solution can be found for the one-operation job crane scheduling problem in linear time [3].

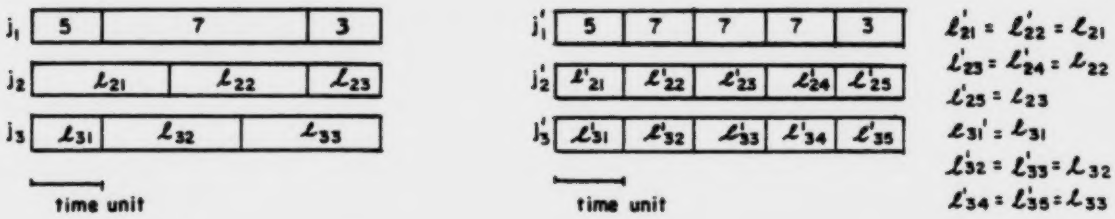
A more complex problem is that where each job has two operations and the operation processing times are all equal. This problem also is NP-complete. Two procedures called batching and meshing are given [4] and the meshing procedure is designed for the scheduling of two cranes, i.e. $\mu = <2,N,(n,2),0,\ell_{ji},\tau>$. For the two crane system, some mixed procedures are defined and combined to form an efficient algorithm whose solution is no worse than 4/3 the optimal solution [4].

When the operation processing times are not equal, formation of the batches, (i.e. the groups of jobs which can be done simultaneously) is a difficult problem. We will confine our attention to the circumstances with equal job processing times, but the number of operations per job and the individual operation processing times are arbitrary, i.e. $\mu = <m,N,(n,n_j),0,\ell_{ji}, p_j = \sum p_{ji} = \text{constant}>$. We will introduce a concept called job compatibility and establish some graph theory models, then, give the CCH method to obtain a schedule by constructing batches with proper sizes. If need be, we can still improve the schedule with the procedure described in section 4.6. All our algorithms are polynonial time.

3.2 Job Compatibility and Comparability Graph Model

The main difficulties in solving crane scheduling problems stem from the interference of cranes resulting in delays. To obtain a good schedule with a lower makespan, we must avoid crane interferences. Therefore, we should concentrate our attention on finding the jobs which can be done simultaneously.

In general, both the number of operations per job and the processing time per operation are not the same. We will introduce the unitary job in which every operation has the same processing time of 1 unit such that a job j with a total processing time p_j, would be a p_j-operation job. Examples are shown in Fig. 7.


j_1
5
7
3
j_2
ℓ_{21}
ℓ_{22}
ℓ_{23}
j_3
ℓ_{31}
ℓ_{32}
ℓ_{33}
time unit
j'_1
5
7
7
7
3
j'_2
ℓ'_{21}
ℓ'_{22}
ℓ'_{23}
ℓ'_{24}
ℓ'_{25}
j'_3
ℓ'_{31}
ℓ'_{32}
ℓ'_{33}
ℓ'_{34}
ℓ'_{35}
time unit
$\ell'_{21} = \ell'_{22} = \ell_{21}$
$\ell'_{23} = \ell'_{24} = \ell_{22}$
$\ell'_{25} = \ell_{23}$
$\ell'_{31} = \ell_{31}$
$\ell'_{32} = \ell'_{33} = \ell_{32}$
$\ell'_{34} = \ell'_{35} = \ell_{33}$

The jobs

The unitary jobs

Fig. 7

For the crane scheduling problems with equal job length p we introduce the following concepts:

Definition 3.1 Relation $\overset{j}{\sim}$ and Job Compatibility Graph.

The relation $\overset{j}{\sim}$ is a binary relation defined on job set J. If j_1, $j_2 \in J$ are two jobs, we denote $j_1 \overset{j}{\sim} j_2$, if the corresponding unitary jobs j'_1, j'_2 meet the condition: if there exists i_0 such that $\ell'_{j_1,i_0} < \ell'_{j_2,i_0}$ (or $\ell'_{j_1 i_0} > \ell'_{j_2,i_0}$), then for all $i = 1 \ldots p$,

$$\ell'_{j_1,i} < \ell'_{j_2 i} \text{ (or } \ell'_{j_1,i} > \ell'_{j_2,i}) \tag{5}$$

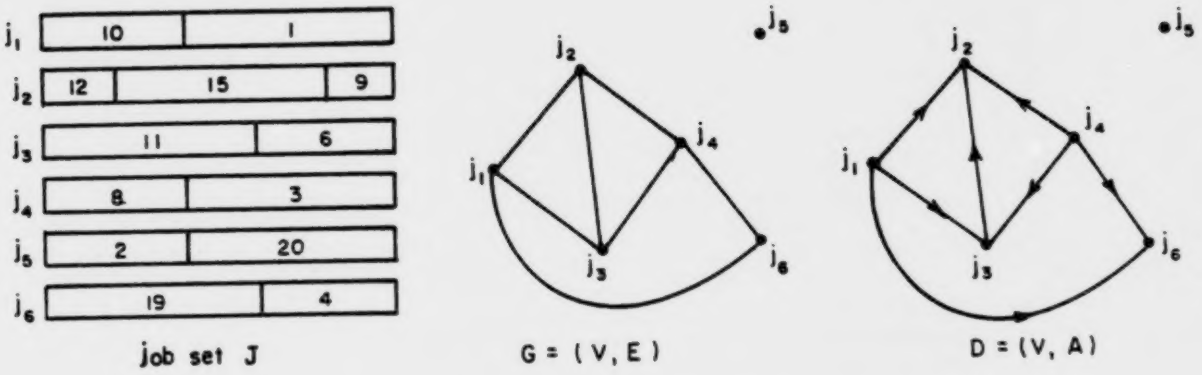
If $j_1 \overset{j}{\sim} j_2$ (simply, denoted by $j_1 \sim j_2$), we can also call jobs j_1, j_2 compatible. Since relation $\overset{j}{\sim}$ possesses symmetry, it can be represented as an undirected graph $G = (V,E)$, called job compatibility graph, for which $u,v \in V$, $uv \in E$ if and only if $u \sim v$ (uv is an edge joining vertices u,v).

Definition 3.2 Relation $\overset{j}{\rightarrow}$ and Job Compatibility Digraph

We define relation $\overset{j}{\rightarrow}$ on set J as follows: if jobs j_1, j_2 are compatible and the corresponding unitary jobs j'_1, j'_2 meet inequalities $\ell'_{j_1,i} < \ell'_{j_2,i}$ $(i=1,\ldots p)$, then, j_1, j_2 have relation $j_1 \overset{j}{\rightarrow} j_2$ (simply, denoted by $j_1 \rightarrow j_2$).

Relation $\overset{j}{\rightarrow}$ on J can be represented as a directed graph $D = (V,A)$ called job compatibility digraph, for which $u, v \in V$, $(u,v) \in A$ if and only if $u \rightarrow v$ ((u,v) is an arc directed from u to v).

The undirected graph G and the directed graph D are related, (i.e. G is the undirected graph of D, and D is an orientation of G). Figure 8 shows a job set and the corresponding compatibility graph G and compatibility digraph D.



job set J

G = (V, E)

D = (V, A)

Fig. 8

Obviously relations $\overset{j}{\sim}$ and $\overset{j}{\rightarrow}$ are transitive, i.e. $j_1 \sim j_2$ and $j_2 \sim j_3 \Rightarrow j_1 \sim j_3$, and $j_1 \rightarrow j_2$ and $j_2 \rightarrow j_3 \Rightarrow j_1 \rightarrow j_3$. However, $\overset{j}{\rightarrow}$ is not symmetric.

Because relation $\overset{j}{\rightarrow}$ is transitive, job compatibility graph G is a comparability graph (in fact, job compatibility digraph D is a transitive orientation), for crane scheduling problems with equal processing times. By the way, we point out that the compatibility digraph is always acyclic (see Fig. 8, in which D has no directed cycle) and the compatibility graph may be a chordal graph which we will discuss in detail in 3.4.

The comparability graphs are perfect graphs and have many important properties. As we know, the problems of finding a Hamiltonian path, of finding a maximum independent set and so on are all NP-complete problems for general graphs. However, for comparability graphs, we can solve the following problems in polynomial time (see Appendix B):

the Hamiltonian path problem,
the clique cover number problem,
the maximum clique problem, and
the independence number problem.

These properties are very useful for crane scheduling problems with equal processing times.

3.3 Some Sufficient and/or Necessary Conditions and the Lower Bound of Makespan

According to the definition of job compatibility, we know that a group of jobs can be done simultaneously iff each pair of jobs is compatible. The jobs of the group will certainly finish simultaneously if they started together, because all jobs are assumed to have equal processing time. If we can partition the job set J into $\lceil \frac{n}{m} \rceil$ disjoint subsets of which every subset is such a compatible group, then, we will obtain an optimal schedule with a makespan $\lceil \frac{n}{m} \rceil \cdot p$, where p is the job processing time, because the total idletime of the schedule is minimum. Such a schedule will be denoted by S^*.

We can introduce the following theorem:

Theorem 3.1. J is the job set with equal job processing time p, $G = (V,E)$ is the compatibility graph, $D = (V,A)$ is the compatibility digraph. m is the number of cranes, n is the number of jobs. We have:

(1) A subset $J' \subseteq J$, $j_i \in J'$, $i = 1,2 \ldots |J'|$ can be done simultaneously

iff j_i $(i=1,2 \ldots |J'|)$ belong to the same clique C of G;

iff j_i $(i=1,2 \ldots |J'|)$ belong to the same directed path of D;

(2) A necessary condition for the existence of an optimum schedule S^* is that the clique cover number K(G) of G satisties $K(G) < \lceil \frac{n}{m} \rceil$;

A sufficient condition for the existence of a schedule S^* is that D contains a directed Hamiltonian path;

A necessary and sufficient condition for the existence of a schedule S^* is that there exists a clique cover of size $\lceil \frac{n}{m} \rceil$ in G with maximum clique size $< m$;

(3) Assume that the independence number of G is $\alpha(G)$, a lower bound of the makespan can be obtained below:

$$\bar{M} = \max\{ \lceil \tfrac{n}{m} \rceil, \alpha(G)\} \cdot p \qquad (6)$$

(4) If the number of cranes m has not been specified, then the minimum makespan which could be achieved is $K(G) \cdot p$. Suppose that clique, minimum clique cover and the collection of minimum clique covers are denoted by C, Q and S(Q) respectively, then the minimal number of cranes needed to achieve the optimal schedule with makespan $K(G) \cdot p$ is:

$$m' = \min_{Q \in S(Q)} \max_{C \in Q} |C| \qquad (7)$$

(5) (1)-(3) above can all be examined or calculated in polynomial time, except the "necessary and sufficient" condition in (2).

Discussion (1). We only need to show

Jobs j_i, j_k $(\forall j_i, j_k \in J')$ are compatible
iff $ji(\forall j_i \in J')$ belong to the same clique C of G;
iff j_i $(\forall j_i \in J')$ belong to the same directed path of D.

Assume that $j_i \sim j_k$ ($\forall\, j_i, j_k \varepsilon J'$), and the vertices corresponding to j_i, j_k in $G = (V,E)$ are v_i, v_k, respectively. Then, edge $v_i v_k \varepsilon E$, $\forall\, v_i, v_k \varepsilon C$, where $C \subseteq V$ consists of the vertices corresponding to J', i.e. C is a clique of G. Conversely, if C is a clique of G, then $v_i v_k \varepsilon E$, $\forall\, v_i, v_k \varepsilon\, C$, i.e. $j_i \sim j_k$, $\forall\, j_i, j_k \varepsilon J'$.

Now assume that $j_i \sim j_k$ $\forall\, j_i, j_k\, \varepsilon J'$. By inequalities (5) in 3.2, we have $\ell'_{ji,i} < \ell'_{jk,i}$ or $\ell'_{jk,i} < \ell'_{ji,i}$ $(i=1,2,\ldots p)$, for any two jobs $j_i, j_k \varepsilon J'$. These conditions imply that all jobs of J' can be ranked in increasing order of their processing locations. The resulting sequence might be $j_1, j_2 \ldots j_{|J'|}$. According to the definition of the relation $\overset{j}{\rightarrow}$, we have $j_1 \rightarrow j_2 \rightarrow \ldots \rightarrow j_{|J'|}$, that is, all vertices corresponding to the jobs belonging to J' are in the same directed path of $D = (V,A)$. Conversely, assume the jobs of J' belonging to the same directed path $P = \ldots j_1 j_2 \ldots j_{|J'|} \ldots$ (i.e. $j_1 \rightarrow j_2 \rightarrow j_{|J'|}$) by the transitivity of relation $\overset{j}{\rightarrow}$, we conclude that the corresponding vertices $v_1, v_2 \ldots v_{|J'|}$ form a complete directed subgraph of D and a complete subgraph of G (i.e. subset $C = \{v_1, v_2 \ldots v_{|J'|}\}$ is a clique of G). By the proof previously given, $j_i \sim j_k$, $\forall\, j_i, j_k \varepsilon J'$.

(2) The necessary condition is obvious. We know that it is needed to partition set J into at least $K(G)$ subsets for each of which the jobs can be done simultaneously according to the definition of $K(G)$ and proof above. Hence, $K(G) < \lceil \frac{n}{m} \rceil$.

Assume that D contains a directed Hamiltonian path P. By the necessary and sufficient condition in (1), existence of P means that all jobs are compatible to each other. So, we can partition them arbitrarily into $\lceil \frac{n}{m} \rceil$ groups of sizes less than or equal to m, and rank the groups in any order. The resulting schedules will all have a makespan $\lceil \frac{n}{m} \rceil \cdot p$.

Obviously, the necessary and sufficient is valid too.

(3) According to the definition of an independent set in the associated job compatability graph G, any two vertices (jobs) belonging to the same independent set have no path joining them, i.e. they cannot be done simultaneously. There are at least $\alpha(G)$ such jobs. It follows that equation (6) is valid.

(4) If m has not been specified, we can possibly obtain a schedule with makespan $K(G)\cdot p$ and there can be no schedule with a smaller makespan, as mentioned in (2) of the discussion. There may be more than one minimum clique cover, and the maximum sizes of cliques belonging to different minimum clique covers may be not the same. In other words, we can use different numbers of cranes to process the jobs and the makespans will all be $K(G)\cdot p$. Therefore, equation (7) is valid obviously.

(5) The conditions in (1) can all be examined in polynomial time. In fact, when graph G and digraph D are represented by their adjacency matrices (or by the incidence matrices), it requires at most $|J'|\cdot(|J'|-1)$ judgements to determine if j_i $(i=1,2,\ldots|J'|)$ is a clique of G. In examining the second condition, when $J' = J$, namely $|J'|=|J|=|V|$, it is equivalent to the directed Hamiltonian path problem. Examining the sufficient condition (2) is also the same problem and it is normally NP-complete. However, we mentioned that the compatibility digraph D is acyclic, since the relation $\overset{j}{\to}$ is transitive. Because the Hamiltonian path problem can be solved in polynomial time for an acyclic digraph [9], it follows that the conditions can be examined in polynomial time.

To examine the necessary condition and calculate the makespan lower bound, we need to calculate the clique cover number K(G) and independence number $\alpha(G)$. For the compatibility graph G, K(G) and $\alpha(G)$ can be calculated in polynomial time. In fact, we have $K(G) = \alpha(G)$, because G is perfect graph [10]. Since D is acyclic, there is at least one vertex called a source and one vertex called a sink in D [9]. A minimum flow problem can be constructed for the problem of finding K(G) or $\alpha(G)$ and K(G) or $\alpha(G)$ can be determined by solving that flow problem [12]. The minimum flow problem is polynomial time solvable [11].

We cannot yet find the minimum clique cover and maximum independent set for a comparability graph in polynomial time. So, we do not know if the necessary and sufficient condition in (2) and m' in (4) of the theorem can be examined or calculated in polynomial time.

Now, we will illustrate the theorem by an example. Assume that $m = 2$, $n = 7$, $p = 3$, and the jobs are given by Table 2.

<table>
<tr><td>j_1</td><td>3</td><td>7</td><td>5</td></tr>
<tr><td>j_2</td><td colspan="2">2</td><td>13</td></tr>
<tr><td>j_3</td><td colspan="2">10</td><td>8</td></tr>
<tr><td>j_4</td><td>15</td><td colspan="2">20</td></tr>
<tr><td>j_5</td><td>14</td><td>4</td><td>19</td></tr>
<tr><td>j_6</td><td>12</td><td>11</td><td>16</td></tr>
<tr><td>j_7</td><td>1</td><td>3</td><td>18</td></tr>
</table>

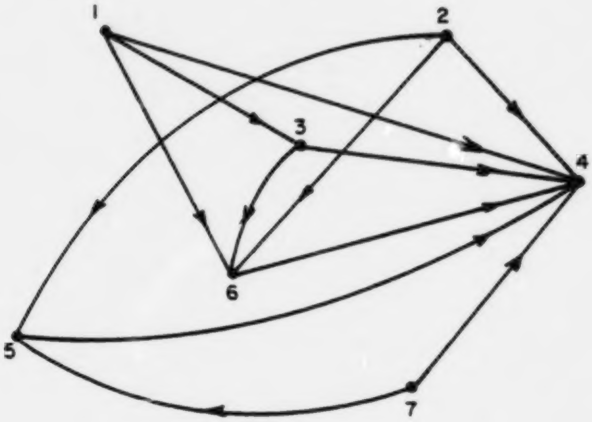
j_1'	3	7	5
j_2'	2	2	13
j_3'	10	10	8
j_4'	15	20	20
j_5'	14	4	19
j_6'	12	11	16
j_7'	1	3	18

time unit

The jobs The unitary jobs

Table 2

The compatibility digraph D and graph G are shown in Fig. 9 and Fig. 10, respectively.



directed paths of length ≥ 2

1 3 6 4

1 3 6

2 5 4

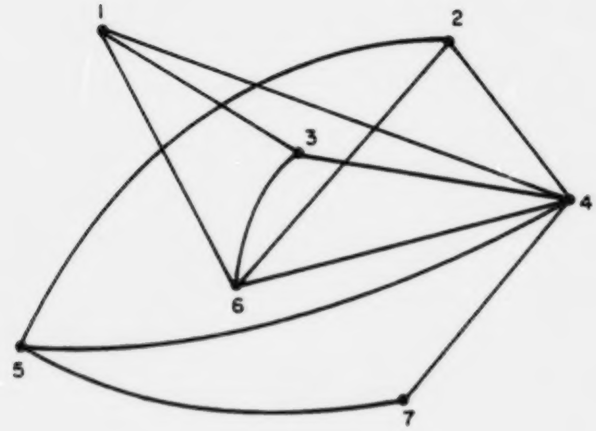
2 6 4

3 6 4

7 5 4 etc.

no directed Hamiltonian path.

Fig. 9



minimum clique covers

$\{1,3,4,6\}, \{5,7\}, \{2\}.$

$\{1,3,6\}, \{2,5,4,\}, \{7\}$

$\{1,3,6\}, \{4,5,7,\}, \{2\}$

clique cover number $K(G) = 3$

maximum independent sets

$\{1,2,7,\}$

$\{3,2,7\}$

independent number $\alpha(G) = 3$

Fig. 10

All the conclusions of the theorem are easy to test and verify. For example, schedule S* exists, because there are clique covers of size $\lceil \frac{n}{m} \rceil = 4$ (say, $\{1,3\}$, $\{4,6\}$, $\{5,7\}$, $\{2\}$). If m is specified as 2, we have a lower bound of the makespan $\bar{M} = 4p$; actually, $\bar{M}$ also is the minimum makespan. If m can be increased, we can obtain a schedule with makespan 3p and m' = 3 cranes are used. Some S* and the schedule S with makespan 3p are shown in Figs. 11-13 along with other schedule S' with a makespan 3p and 4 cranes.

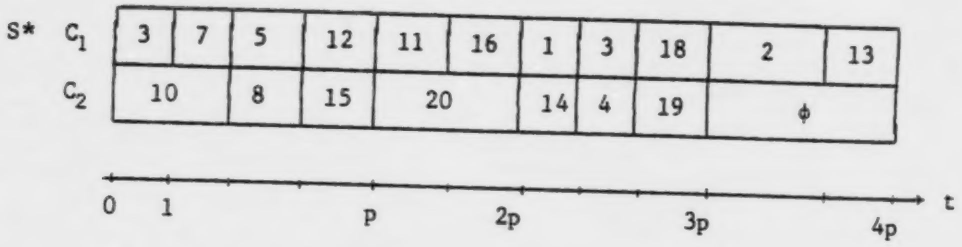

S*
C_1
3
7
5
12
11
16
1
3
18
2
13
C_2
10
8
15
20
14
4
19
ϕ
0
1
p
2p
3p
4p
t

Fig. 11

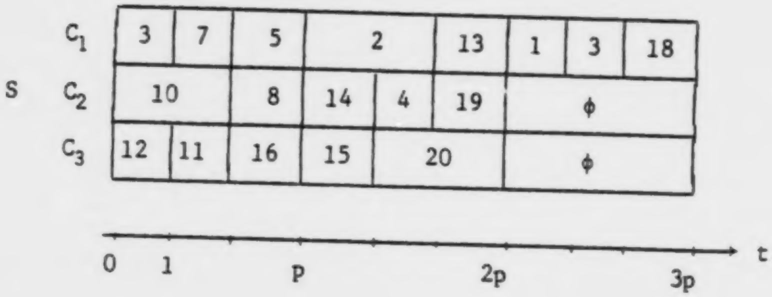

C_1
3
7
5
2
13
1
3
18
S
C_2
10
8
14
4
19
ϕ
C_3
12
11
16
15
20
ϕ
0
1
P
2p
3p
t

Fig. 12

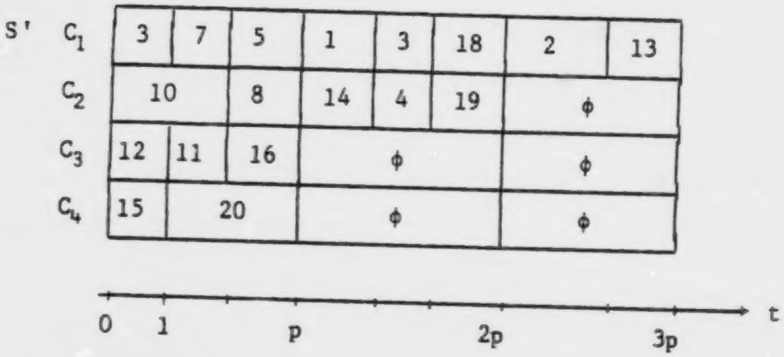
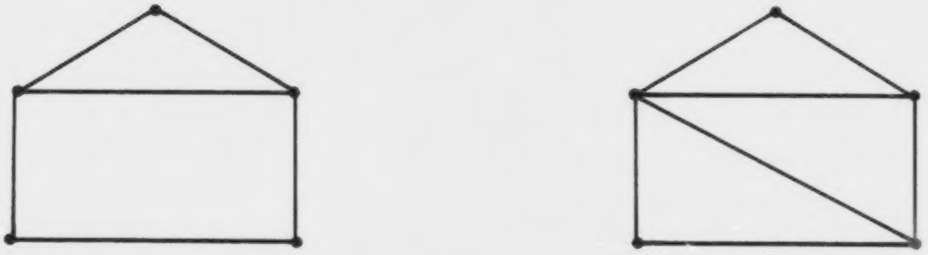

S'
C_1
3
7
5
1
3
18
2
13
C_2
10
8
14
4
19
ϕ
C_3
12
11
16
ϕ
ϕ
C_4
15
20
ϕ
ϕ
0
1
p
2p
3p
t

Fig. 13

3.4 Chordal Graph Approximate Model

In section 3.2, we have established the job comparability graph model for the crane scheduling problem with equal job processing times. The comparability graphs are perfect, hence, $K(G) = \alpha(G)$ for each comparability graph G, and their values can be obtained in polynomial time.

Because a comparability graph may or may not be a chordal graph, the job compatibility graph $G = (V,E)$ may or may not be a chordal graph. The chordal graph has some more useful properties than the comparability graph. For example, for the chordal graph, not only the problems of finding $K(G)$ or $\alpha(G)$ but also the problems of finding the minimum clique cover and maximum independent set are solvable in polynomial time. In view of these facts, if $G = (V,E)$ is not a chordal graph we hope to obtain a chordal graph $G' = (V,E')$ by adding a minimum number of edges to G. In doing so, the resulting chordal graph $G' = (V,E')$ will be an approximation of the job compatibility graph G which we will call a chordal graph approximate model of the job compatibility graph. We can use the ratio β of the number of added edges to the edge number of G to express how closely the model G' approximates the job compatibility graph G, i.e., measuring the approximation by $\beta = (|E|'-|E|)/|E|$. For example, for G and G' in Fig. 14 we have $\beta = 1/6$.



G

G'

Fig. 14

We know that the job compatibility graph G = (V,E) for a crane scheduling problem is a comparability graph and the job compatibility digraph D = (V,A) is a transitive orientation of G. Now, we present the following problems: (1) If the approximate model G' = (V,E') is a chordal graph, is it also a comparability graph? (b) If it is, is there a transitive orientation D' of G' such that D $\subseteq$ D', i.e., D is a subgraph of D'? We first introduce the following concepts, before answering the questions:

Definition 3.1 The Triangulated Digraph of the Job Compatibility Diagraph D

We say that a digraph D' = (V,A') is a triangulated digraph of D = (V,A), if D' satisfies

(1) D' is an orientation of G' = (V,E');
(2) D $\subseteq$ D'.

There may exist more than one triangulated digraph for a given job compatibility digraph D.

The questions presented previously are significant for the purpose of introducing the approximate model. If the answers are positive for both of these, then, groups of jobs which can be done simultaneously, can be constructed in the following two ways.

1. First, find the minimum clique cover of G'. Then for each oriented subgraph induced by a clique, find a Hamiltonian directed path.

2. Find the longest path in the triangulated digraph D'.

We know that, because such a subgraph is transitive, it has only one Hamiltonian directed path which can be found in polynomial time (see Appendix B). Since D' becomes an acyclic digraph, the longest directed path problem is polynomial (see Appendix B).

However, we will soon see that the answer to the first question is not positive (i.e. G' may or may not be a comparability graph), but the comparability graph recognition algorithm is polynomial [26]. If G' is shown to be a comparability graph by the recognition algorithm, then, we should be able to answer the second question. In fact, if the answer is negative, then the answer to the first question is of no importance.

Is G' a comparability graph? Figure 15 shows a comparability graph G and the corresponding digraph D. An approximate model G' and its transitive orientation D' are shown in Fig. 16. For this example, G' is a comparability graph.

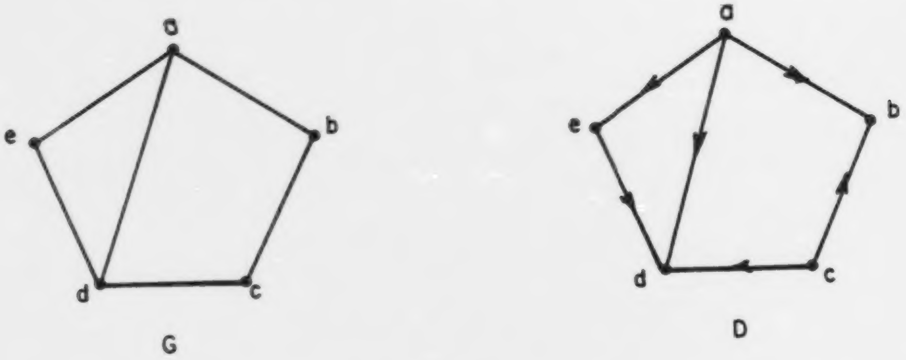


Fig. 15

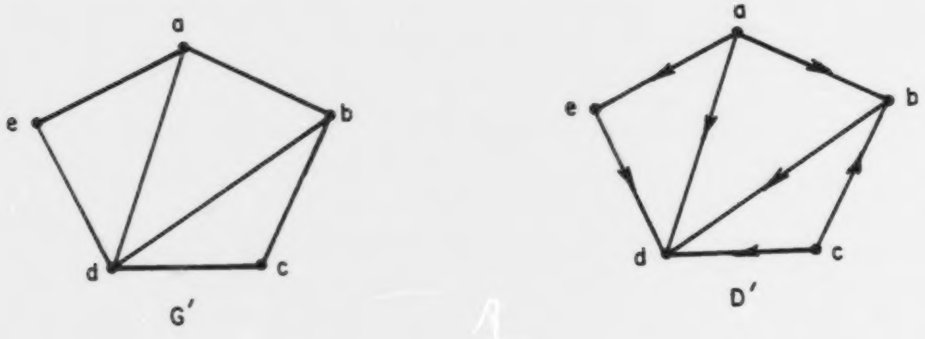


Fig. 16

Another example is shown in Figs. 17, 18, 19. For G and D in Fig. 17, there are two possible approximate models, the one in Fig. 18 being a comparability graph, but another (in Fig. 19) is not, because there is an odd closed walk (see TRO Theorem. Appendix B). For G and D in Fig. 20, each approximate model is not a comparability graph (see Fig. 21).

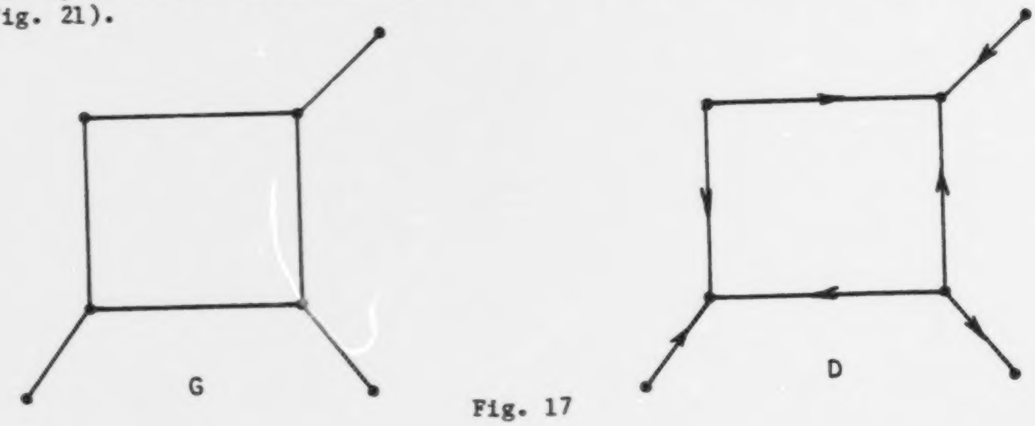
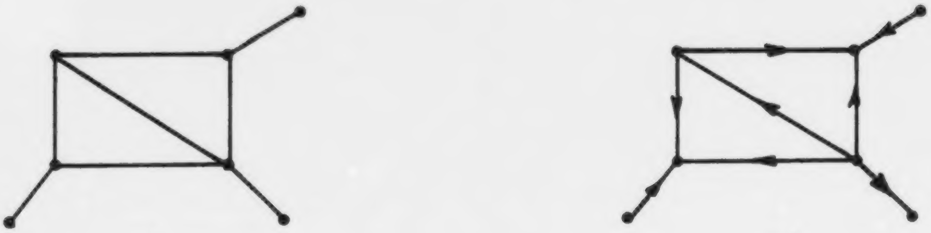
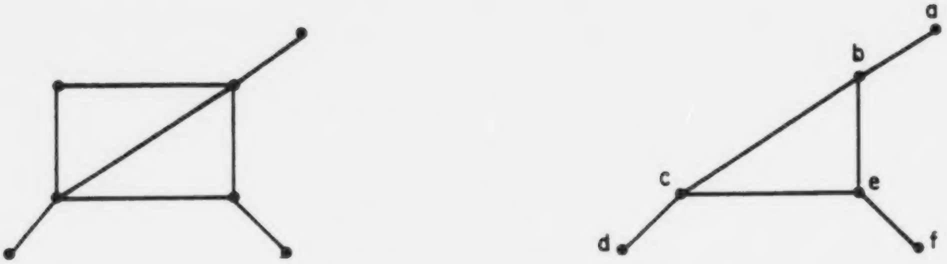


Fig. 17



G′ D′

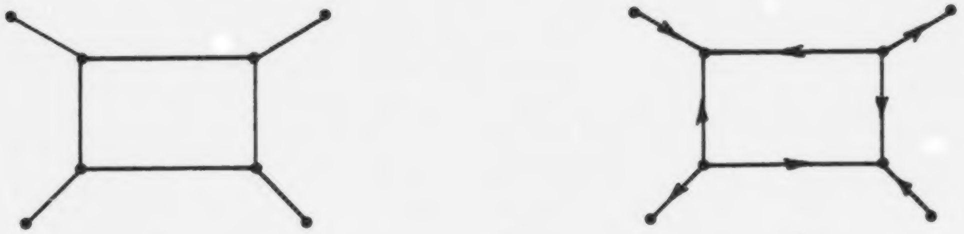
Fig. 18



G′

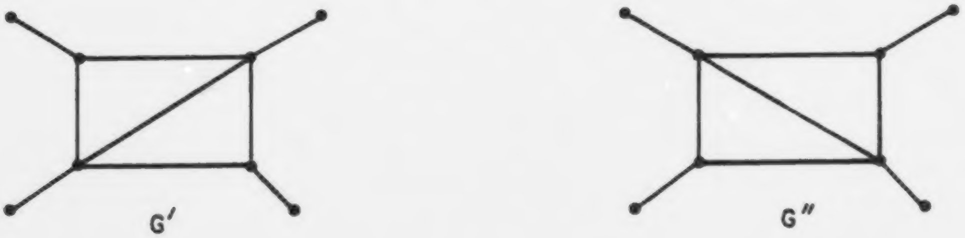
an odd closed walk : abcdcefeba
the walk is nonchordal

Fig. 19



G D

Fig. 20



G′ G″

Fig. 21

Assume that we are given a G-decomposition of G and a decomposition scheme δ (for the definitions, see Appendix B). The following algorithm can be applied to construct a triangulated digraph.

Algorithm 3.1 (Triangulated Digraph Algorithm)

Input Job compatibility graph $G = (V,E)$; job compatibility digraph $D = (V,A)$. G-decomposition of G: $E = E_1 + \ldots + E_K$; decomposition scheme $\delta = \{(x_1,y_1)\ldots(x_K,y_K)\}$; $B = \{B_1 \ldots B_K\}$, $B_i = \{(a_{i1},b_{i1}),\ldots,(a_{i|B_i|},b_{i|B_i|})\}$ for $i = 1,2,\ldots K$; chordal graph approximate model $G' = (V,E')$.

Output K', δ', B' for G'.

```
Steps
1. Initialize: $i \leftarrow 1$, $j \leftarrow 1$, $K' \leftarrow 0$, $\delta' \leftarrow B' \leftarrow \phi$, $E' \leftarrow E'$;
2. Arbitrarily pick $e = (x_{ij},y_{ij}) \in B_i$, $\delta' \leftarrow \delta' \cup e$;
3. Enumerate the implication class $B_{ij}$ of $E'$ containing e;
4. $E' \leftarrow E' - (B_{ij} \cup B_{ij}^{-1})$, $B_i \leftarrow B_i - (B_{ij} \cap E)$;
5. $K' \leftarrow K' + j$, $B' = B' \cup B_{ij}$; If $E' = \phi$, stop;
   Otherwise, if $i < K$, go to 5.1; otherwise, go to 5.2;
   5.1 If $B_i = \phi$, $i \leftarrow i + 1$, $j \leftarrow 1$, go back to 2;
   Otherwise, $j \leftarrow j + 1$, go back to 2;
   5.2 $j \leftarrow 1$; arbitrarily pick $e = (x_{ij},y_{ij}) \in E'$;
   Enumerating the implication class $B_{ij}$ of $E'$ containing e;
   $E' \leftarrow E' - B_{ij}$; go back to 5.
```

Let $\delta = ((x_1,y_1))\ldots(x_K,y_K))$ be a decomposition scheme of G and $B = \{B_i,\ i=1,\ldots K\}$ be the collection of the implication classes containing (x_i,y_i), $(i=1,\ldots K)$ respectively. For $G' = (V,E')$, the corresponding scheme and collection are respectively, $\delta' = ((x_{11},y_{11}), \ldots, (x_{ij},y_{ij}), \ldots, (x_{K1},y_{K1}), \ldots, (x_{Kj_K},y_{Kj_K}), \ldots, (x_{K+1,1},y_{K+1,1}), \ldots, (x_{K'1},y_{K'1}))$ and $B' = \{B_{11},\ldots,B_{1j_1},\ldots,B_{Kj_K}, B_{K+1,1},\ldots B_{K'1}\}$, obtained by Algorithm 3.1.

Theorem 3.1. Assume that $G = (V,E)$ and $D = (V,A)$ are job compatibility graph and digraph, respectively. $G' = (V,E')$ is the chordal graph approximate model of G. If G' is a comparability graph, and $B_i \in B$, $B_{ij} \in B'$ obtained by Algorithm 3.1 satisfy the following conditions

$$(B_{ij} \cap E) \subseteq \textstyle\sum_{\ell=1}^{K} B_i, \quad i=1,\ldots K, \quad j=1,\ldots j_i \tag{9}$$

then there exists a transitive triangulated digraph $D' = (V,A')$ of D.

Proof. By the hypothesis of the theorem, G' is a comparability graph. From the TRO theorem we have $B_{ij} \cap B_{ij}^{-1} = \phi$ for each $B_{ij} \in B$. It implies that digraph $D = (V,A')$, where $A' = B_{11} + \ldots + B_{1j_1} + \ldots + B_{K1} + \ldots + B_{Kj_K} + B_{k+1,1} + B_{K'1}$, is a transitive orientation of G'. From Algorithm 3.1, we know that $(B_{K+1,1} \cup \ldots \cup B_{K'1}) \cap E = \phi$. Hence

$$\begin{aligned}&A' \cap E \\ &= (B_{11}+\ldots+B_{Kj_K} + B_{K+1,1}+\ldots B_{K'1}) \cap E \\ &= B_{11} \cap E + \ldots + B_{Kj_K} \cap E \\ &\subseteq B_1 + \ldots + B_K \quad \text{(by (9))} \\ &= A\end{aligned}$$

i.e, $A' \cap E \subseteq A$ (10)

On the other hand, according to Algorithm 3.1, each arc $f \in B_i$, either is included by $\{(x_{i1},y_{i1}),\ldots,(x_{ij_i},y_{ij_i})\}$ or belongs to a B_{ij} $j \in \{1,2,\ldots j_i\}$. It follows that

$$(B_i+\ldots+B_{ij_i}) \cap E \supseteq B_i; \quad i = 1,\ldots K \tag{11}$$

Hence

$$A' \cap E \supseteq A \tag{12}$$

From (10) and (12), we have

$$A' \cap E = A \tag{13}$$

Equation (13) means that $D = (V,A) \subseteq D' = (V,A')$ (in general, $\subset$ holds), namely, D' is a triangulated digraph of D, moreover, D' is a transitive triangulated digraph.

Q.E.D.

We illustrate the theorem by an example. See Figs. 22, 23. D has only one implication class

$$B_1 = \{(a,e),(a,b),(a,d),(c,b),(c,d),(e,d)\}$$

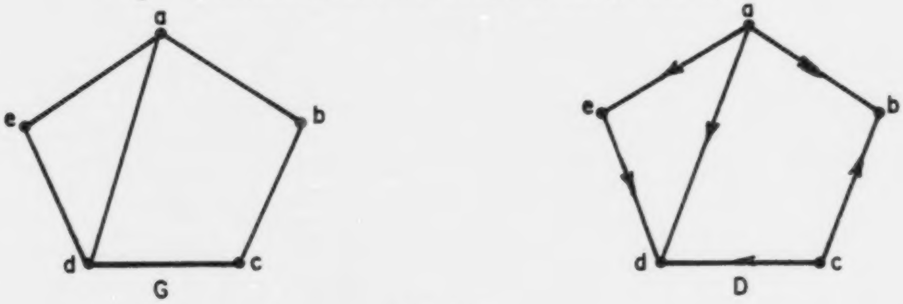


Fig. 22

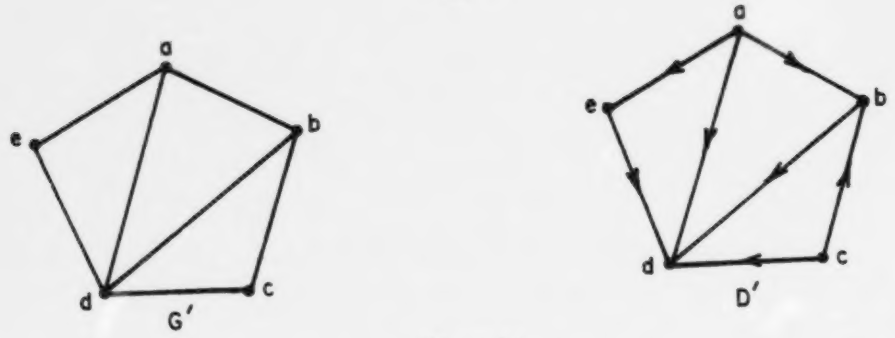


Fig. 23

Arbitrarily select

$$(x_{11}, y_{11}) = (a,e) \varepsilon B_1$$

then

$$B_{11} = \{(a,e),(a,b),(c,b)\}$$

$$B_1 - (B_{11} \cap E) = \{(a,d),(c,d),(e,d)\}$$

$$E' - (B_{11} \cup B_{11}^{-1}) = \{(a,d),(d,a),(c,d),(d,c),(e,d),(d,e),(d,b),(b,d)\}$$

Select

$$(x_{12}, y_{12}) = (a,d) \varepsilon B_1 - B_{11}$$

then

$$B_{12} = \{(a,d),(b,d),(c,d),(e,d)\}$$

$$B_1 - (B_{11} \cup B_{12}) \cap E = \emptyset$$

$$E' - (B_{11} \cup B_{11}^{-1} \cup B_{12} \cup B_{12}^{-1}) = \emptyset$$

Finally, we have $D' = (V,A')$. $A' = B_{11} \cup B_{12} = \{(a,e),(a,b),(c,b),(a,d), (b,d),(c,d),(e,d)\}$.

At the end of this section, we state the following fact: $K(G) > K(G')$ and $\alpha(G) > \alpha(G')$. $\alpha(G) > \alpha(G')$ is evident because $E \subseteq E'$. Moreover, $K(G) = \alpha(G)$, $K(G') = \alpha(G')$ (G,G' all are perfect), so, $K(G) > K(G')$.

3.5 Nonchordal Circuit Recognition and Elimination

It is possible to find a good schedule for the approximate model $G' = (V,E')$ because G' is a chordal graph. However, G' contains the added edges which are not contained in the job compatability graph

G, therefore, an appropriate procedure for removing the added edges at an appropriate time is needed in order to obtain a good schedule for G.

The CCH method consists of three algorithms which have the following functions:

- To examine if G is a chordal graph;
- If G is not, to add a minimum number of edges such that the resulting graph G' is a chordal graph;
- To find the minimum clique cover of G';
- To find the directed Hamiltonian path of the subgraphs of D' induced by the cliques and construct the schedule.

A chordal graph can be recognized using a linear time algorithm [17]. However, the algorithm is not very helpful, because we not only want to recognize a chordal graph but also to construct one with the minimum number of added edges if the given graph is not a chordal graph. The number of added edges required for forming a chordal graph depends upon which edges are to be added and in what order they are added. An undirected graph G and all the circuits of length strictly greater than 3 are shown in Fig. 24. We can add 1 edge, 2 edges, 3 edges, 4 edges or more than 4 edges to G so that the resulting graph G' is a chordal graph. The cases for adding 1 edge or 2, 3, 4 edges are shown in Fig. 25.

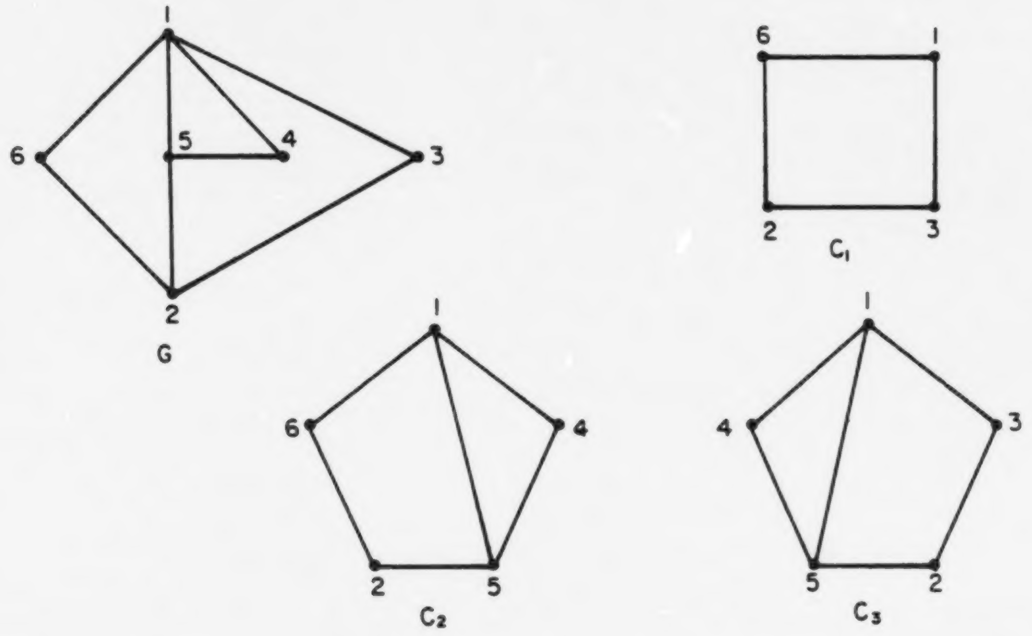
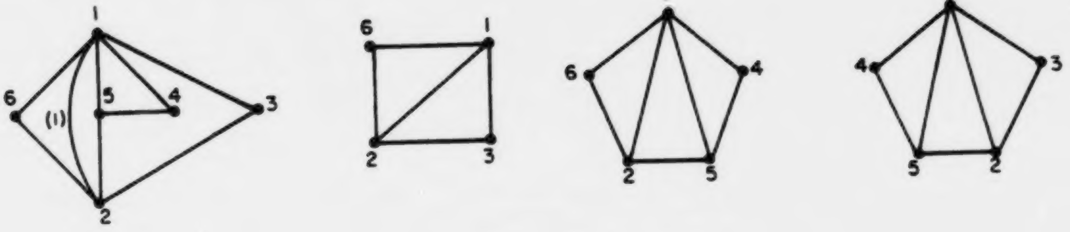
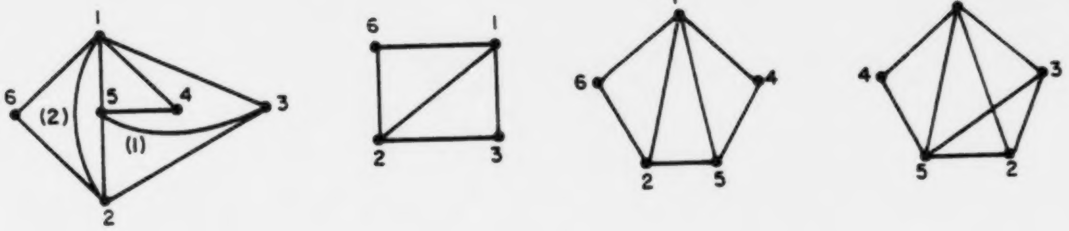


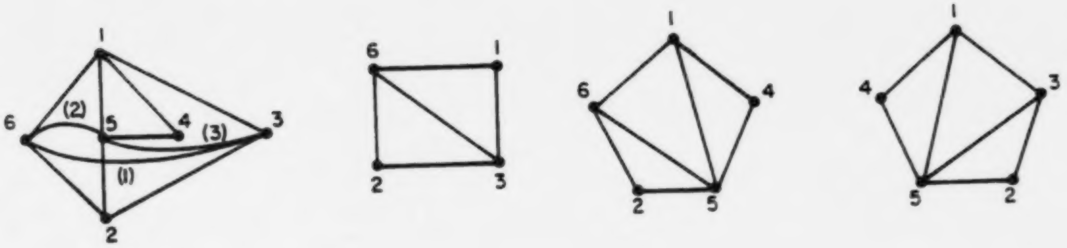
Fig. 24


1
6
5
4
3
2
(1)
6
1
2
3
1
6
4
2
5
1
4
3
5
2

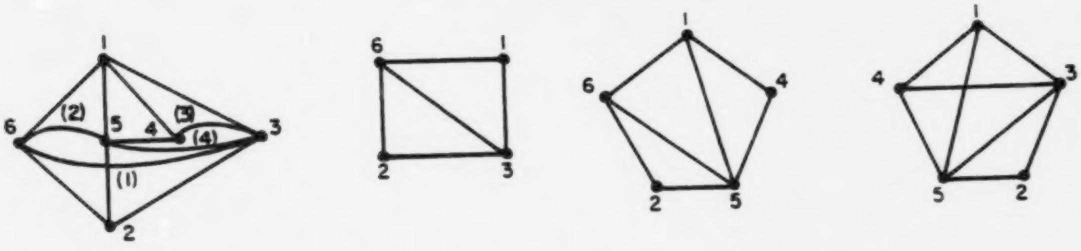
The added edge sequence (12)


1
6
(2)
5
4
3
(1)
2
6
1
2
3
1
6
4
2
5
1
4
3
5
2

The added edge sequence (35, 12)


1
6
(2)
5
4
(3)
3
(1)
2
6
1
2
3
1
6
4
2
5
1
4
3
5
2

The added edge sequence (63,65,53)


1
6
(2)
5
4
(3)
(4)
3
(1)
2
6
1
2
3
1
6
4
2
5
1
4
3
5
2

The added edge sequence (63,65,43,53)

Fig. 25

We call a circuit (path) a *nonchordal* circuit (path), if it does not possess any chord. If a circuit (path) with their chords constitutes a chordal graph, we call it a chordal circuit (path). A circuit $v_0 v_1 \ldots v_\ell v_0$ (or a path $v_0 v_1 \ldots v_\ell$) is chordless if $v_i v_j \bar{\varepsilon} \hat{E}$ for i and j differing by more than 1 mod $\ell + 1$.

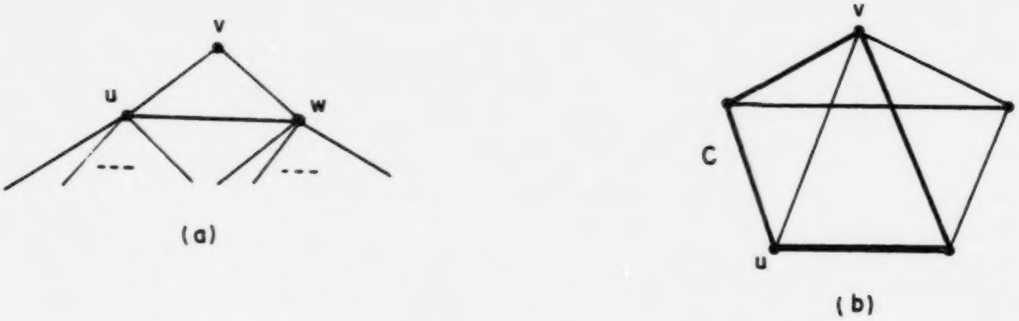
According to the definition of a chordal graph, recognition of a chordal graph involves the question: "Is there any nonchordal circuit of length strictly greater than 3?". If there is, we hope to find all the nonchordal circuits and add the minimum number of edges to them such that there is no chordless circuit. We will develop two algorithms. The first can be used to find all the nonchordal circuits and paths for the given graph. The second will construct a chordal graph.

Lemma 3.2. The following kinds of vertices are not related to the chordality for a given graph $G = (V,E)$:

1. contained by a triangle and the degree being two;
2. the degree being n−1, where $n = |V|$;
3. the pendant vertices, i.e., the vertices of degree 1.

Proof. A vertex of degree 2 and contained by a triangle cannot belong to any nonchordal circuits. Assume v is such a vertex (see Fig. 26a). If $v \varepsilon C$, C is a circuit, then $u, w \varepsilon C$ and edge $u w \varepsilon E$. That is, C has at least one chord uw. If v is the vertex in Fig. 26b and $v \varepsilon C$, then there is a vertex $u \varepsilon C$, u,v being not consecutive and $uv \varepsilon E$, since $d_v = n - 1$. As for the pendant vertices, obviously, the chordality of G has nothing to do with them.

Q.E.D.



(a)

(b)

Fig. 26

By Lemma 3.2, the problem of recognizing chordal graphs can be reduced by removing such vertices. Let G' be the resulting graph, we can go through the same process again and again, until there are no such vertices. Fig. 27 illustrates the application of Lemma 3.2.

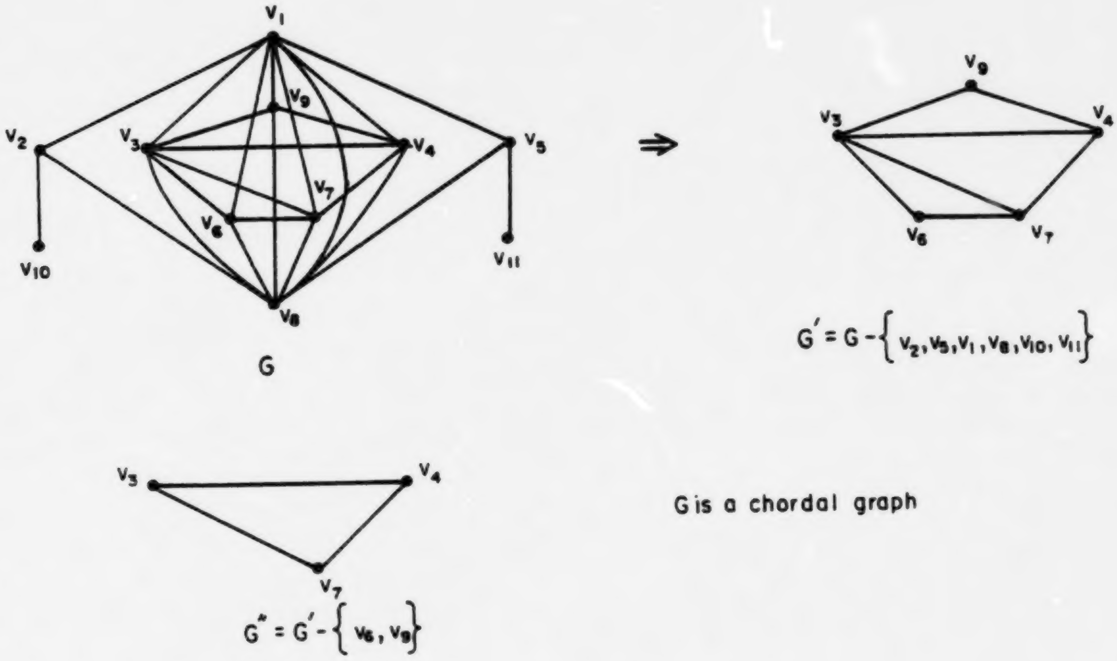

v1
v9
v2
v3
v4
v5
v7
v6
v10
v11
v8
G
⇒
v9
v3
v4
v6
v7
G' = G − {v2, v5, v1, v8, v10, v11}
v3
v4
v7
G'' = G' − {v6, v9}
G is a chordal graph

Fig. 27

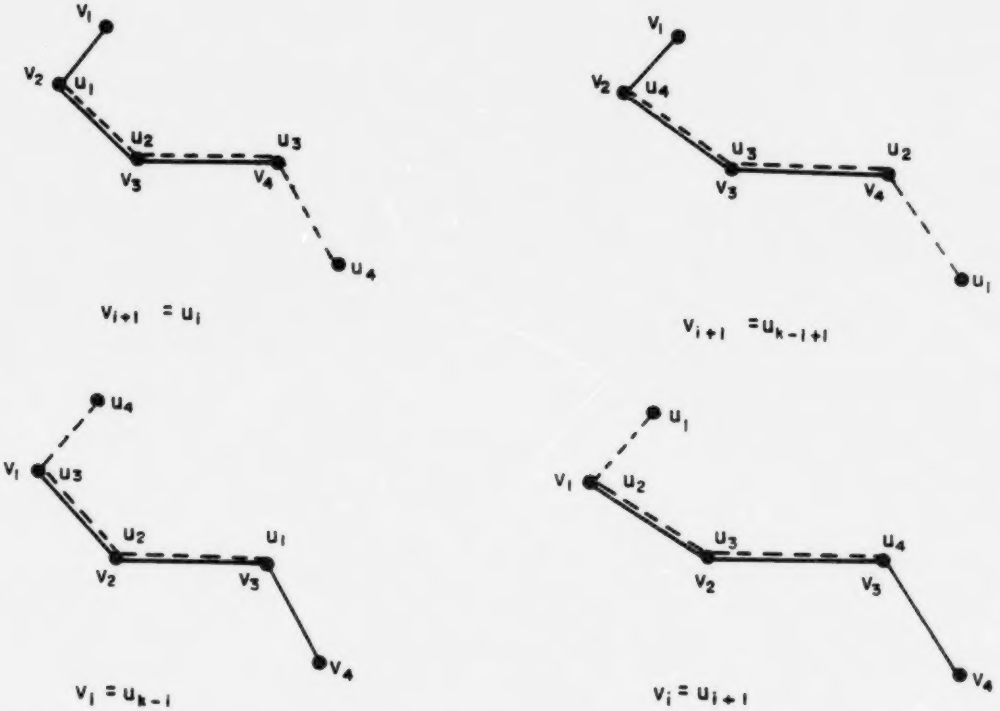
In general, the ultimate graph is not as simple as obtained in Fig. 27. There may be many nonchordal circuits in the graph and they must be found. For the purpose, we introduce the following operation between two paths.

Definition 3.2 Overlap of Paths

Assume that we are given an undirected graph $G = (V,E)$ and two K-paths in G: $P_1(K) = v_1v_2\ldots v_K$, $P_2(K) = u_1u_2\ldots u_K$. The overlap of $P_1(K)$ and $P_2(K)$ (denoted by $P_1(K) \wedge P_2(K)$) is defined as follows:

$$P_1(K) \wedge P_2(K) = \begin{cases} v_1 v_2 \cdots v_K u_K & \text{if } v_{i+1} = u_i \\ v_1 v_2 \cdots v_K u_1 & \text{if } v_{i+1} = u_{K-i+1} \\ u_K v_1 v_2 \cdots v_K & \text{if } v_i = u_{K-i} \\ u_1 v_1 v_2 \cdots v_K & \text{if } v_i = u_{i+1} \\ & (i=1,2,\ldots,K-1) \\ \emptyset & \text{Otherwise} \end{cases} \tag{14}$$

Figure 28 is an illustration of the operation (K=4).


v₁
v₂
u₁
u₂
u₃
v₃
v₄
u₄
v₁
v₂
u₄
u₃
u₂
v₃
v₄
u₁
u₄
v₁
u₃
u₂
u₁
v₂
v₃
v₄
u₁
v₁
u₂
u₃
u₄
v₂
v₃
v₄

$v_{i+1} = u_1$ $\quad$ $v_{i+1} = u_{k-i+1}$

$v_1 = u_{k-1}$ $\quad$ $v_i = u_{i+1}$

Fig. 28

Obviously, two K-paths either cannot be overlaped or the overlap is a (K+1)-path. Note that, it is not necessary that the path be nonchordal.

Now suppose that all the K-paths have been found for a given graph G. Overlapping of a pair of K-paths can produce a (K+1)-path. Assume that N(t) is the number of (K+1)-paths produced at time t. A label $L(P_i(K),t)$ can be defined for the ith K-path $P_i(K)$ at the same time.

Definition 3.3 Label of $P_i(K)$ at Time t

The label $L(P_i(K),t)$ can be specified as follows:

1. $L(P_i(K),t)$ is a $N(t)$-bit Boolean vector, the jth bit corresponds to the jth (K+1)-path (or circuit), $j = 1,2,\ldots,N(t)$
2. The jth bit of $L(P_i(K),t)$ is 1 if and only if the ith K-path can be overlapped with another K-path to produce the jth (K+1)-path (or circuit).

Assume that $P_K = \{p_i(K),\ i=1,2,\ldots,i_K\}$ is the set of K-paths in which any K-path cannot appear more than once. $L(i,t) = L(p_i(K),t)$, e_j is a $N(t)$-bit Boolean vector with 1 only at the jth position, V expresses the logical addition of two Boolean vectors. We have the following algorithm:

Algorithm 3.2.1 (Labelling Algorithm)

```
Steps   1.  $L(i,t) \leftarrow 0,\ \forall\ i = 1,2,\ldots,i_K;\ j \leftarrow 0,\ N(t) \leftarrow 0;$
        2.  Every time a (K+1)-path $p(K+1) = p_{i1}(K) \wedge p_{i2}(K)$
            appears, $j \leftarrow j + 1,\ N(t) \leftarrow N(t) + 1$, and do:

            2.1  if $p(k+1)$ is not a circuit, then
                 $L(i_1,t) \leftarrow L(i_1t)\ V\ e_j,\ L(i_2t) \leftarrow L(i_2t)\ V\ e_j,$
                 $L(i,t) \leftarrow L(i,t),\ i \neq i_1,\ i_2;$
            2.2  If $p(k+1)$ is a circuit, let P be the set of the
                 2K K-paths contained by $C(K+1) = p(K+1)$, then
                 $L(i,t) \leftarrow L(i,t)\ V\ e_j$, if $p_i(K) \in P$,
                 $L(i,t) \leftarrow L(i,t)$, if $p_i(K) \notin P$.
```

The paths in a graph can be divided into two classes, chordal paths and nonchordal paths. We are interested only in the nonchordal path. The nonchordal 3-paths are easily listed for a given graph. In reality, if vertices a, b are adjacent with the same vertex c respectively and a, b are not adjacent, then, acb or bca is a nonchordal 3-path. To avoid duplications of 3-paths, we provide that the first vertex of a 3-path always precedes the third vertex in the vertex sequence $(v_1,v_a,\ldots,v_m)$, $n = |V|$.

We can establish the following algorithm for finding all the nonchordal paths and circuits for a given undirected graph.

Algorithm 3.2 (Algorithm for Finding all the Nonchordal Paths and Circuits)

Input an indirected graph $G = (V,E)$

Output $f = \sigma$, f is a Boolean variable, the collection $C = \{C(K), K = 1,2,\dots,n-1\}$, where $n = |V|$ and $C(K) = \{C_1(K), C_2(K),\dots,C_{iK}(K)\}$

Steps

1. Construct all the nonchordal 3-paths $\{p(3)\}$, any 3-path can only occur once, $C \leftarrow \Phi$;
   $P \leftarrow \{p(3)\}$; $L(p(3)) \leftarrow 0$, $\forall\ p(3) \in P$; $K \leftarrow 3$; $f \leftarrow 0$;

2. Construct $\{p_i(K) \wedge p_j(K)\}$, $\forall\ p_i(K), p_j(K) \in P$, $p_i(K) \neq p_j(K)$ and $L(p_i(K)) \cdot L(p_j(K)) = 0$, where symbol $\cdot$ expresses the logical multiplication of two $N(t)$-bit Boolean vectors and 0 being the null vector;

3. As long as a new $(K+1)$-path $p_i(K+1)$ (or circuit $C_i(K+1)$) occurs, modify immediately the labels of the K-paths by Algorithm 3.2.1 which can be overlapped to create the $(K+1)$-path;

4. $C(K+1) \leftarrow \{C_i(K+1)\}$;
   $P \leftarrow \{p_i(K+1)\} - C(K+1)$;
   $K \leftarrow K + 1$;
   If $P \neq \Phi$, go to 2, otherwise, if $C(K) = \phi$ for same K, $f \leftarrow 1$, otherwise stop.

Example. Consider graph G in Fig. 29. The evolutions of the example are illustrated in Fig. 30.

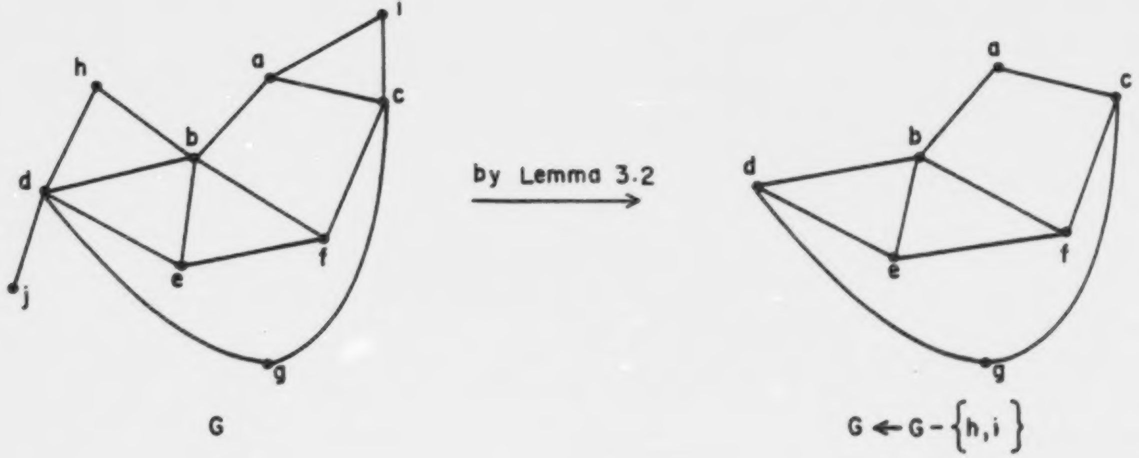


Fig. 29

$L(P_i(3))$						$\{P_i(3)\}$	$\{P_\ell(4) = P_i(3) \wedge P_j(3)\}$	
				100	001	abd	abde	
			1	000	000	abe		
					10	abf	abfc	
					110	acf	acfe	
				11	000	acg	acgd	bacg
			1	110	010	bac	dbac	ebac
			110	000	001	bdg	bdgc	gbdf
		11	000	000	010	bfc	bfcg	dbfc
	1	100	000	000	100	cfe	cfed	efcg
	110	000	010	001	000	cgd	cgde	fcgd
		10	100	000	000	dbf		
1	000	100	000	000	000	def	gdef	
1	010	000	000	000	000	edg		
	101	001	000	000	000	fcg		

$C_1(4) = \text{abfc}$

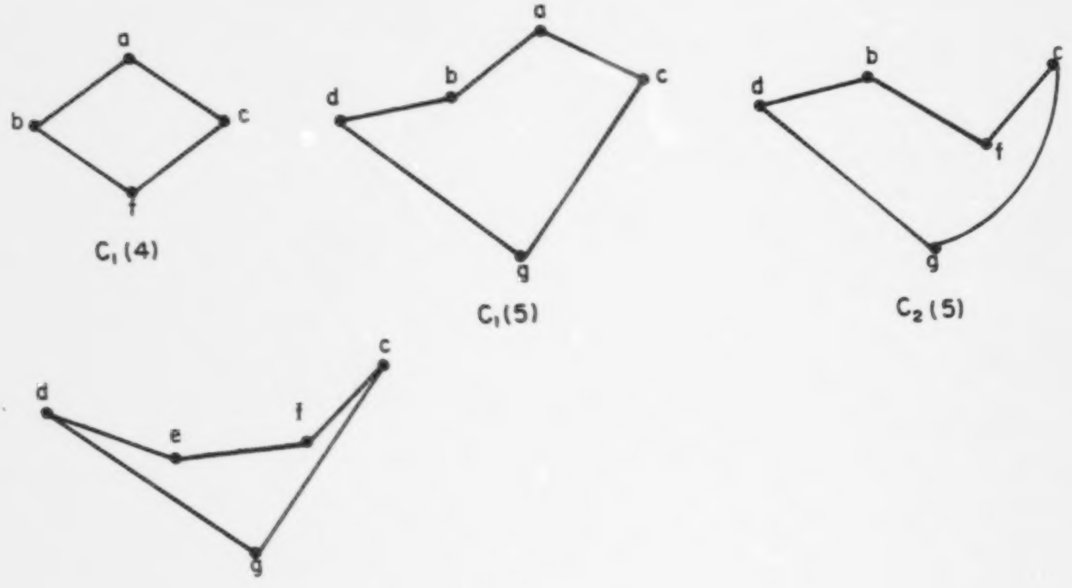
$L(P_i(4))$						$\{P_i(4)\}$	$\{P_\ell(5) = P_i(4) \wedge P_j(4)\}$
					1	abdg	abdgc
					10	acfe	acfed
					101	acgd	acgde
				1	001	bacg	ebacg
					1	dbac	
				1	000	ebac	
				10	000	bdgc	bdgcf
				10	001	gdbf	
				10	000	bfcg	
				10	000	dbfc	
				10	000	cfed	cfedg
				10	000	efcg	
				100	100	cgde	
				110	000	fcgd	
				100	000	gdef	

$C_1(5)$ = abdgc; $C_2(5)$ = bdgcf; $C_3(5)$ = cfedg

$L(P_i(5))$						$\{P_i(5)\}$	$\{P_\ell(6) = P_i(5) \wedge P_j(5)\}$
						acfed	
						acgde	
						ebacg	

There is no nonchordal 6-circuit

All the nonchordal circuits are as follows:



$C_1(4)$ $C_1(5)$ $C_2(5)$

$C_3(5)$

Fig. 30

For the example, acfed, acgde and ebacg are nonchordal paths. Each of them is contained by a chordless circuit: acfed $\subset$ acfedb, acgde $\subset$ acgdeb and ebacg $\subset$ ebacgd. In general, we have the following definition.

Definition 3.4 Longest Nonchordal Paths

Let G be an undirected graph. A nonchordal path in G is called a longest nonchordal path if the following conditions are met:

1. There is no nonchordal circuit which contains it;
2. It is contained by either chordless circuit or a chordless path;
3. There is no nonchordal path longer than it.

The correctness of Algorithm 3.2 should be shown, and we have the following theorem.

Theorem 3.3. G is an undirected graph. σ and C are, respectively, the value of the Boolean variable f and the collection, obtained by Algorithm 3.2 Then the following conclusions are true:

C1. $\sigma = 0$, G is a chordal graph;
C2. $\sigma = 1$, G is not a chordal graph and C is the collection consisting of all the nonchordal K-circuit sets C(K)'s, $K = 1,2,\ldots,n-1$. If there is no nonchordal K-circuit for some K, then $C(K) = \phi$;
C3. Each nonchordal circuit appears once and only once in C;
C4. If $P_{K+1} = \{P(K+1)\} = \phi$ and $P_K = \{P(K)\} \neq \phi$, then $\{P(K)\}$ is the set of the longest nonchordal paths.

Proof. The following facts must be shown first.

1. Any circuit obtained by Algorithm 3.2 is nonchordal. Assume that $P_i(K)$ and $P_j(K)$ are two nonchordal paths. By the definition 3.2, $P(K+1) = P_i(K) \wedge P_j(K)$ is a path having K-1 successive common vertices both with $P_i(K)$ and with $P_j(K)$. Moreover, its one end vertex belongs to $P_i(K)$ but not to $P_j(K)$, the other end vertex belongs to $P_j(K)$ but not to $P_i(K)$. Therefore, P(K+1) must be a nonchordal path. If not, any chord of P(K+1) is also a chord of $P_i(K)$ or/and $P_j(K)$. A contradiction appears. Hence, if P(K+1) is a (K+1)-circuit ($K > 3$), then it must be a nonchordal circuit.

2. Any nonchordal circuit of G can be found by Algorithm 3.2. To show this, it is necessary to show that a (K+1)-path can be split into two K-paths and the (K+1)-path can be produced by overlapping the two K-paths.

Assume that the (K+1)-path is $W_1W_2\ldots W_KW_{K+1}$. Obviously, it can be split into $p_1(K)$ and $p_2(K)$:

$$P_1(K) = \begin{cases} W_1W_2\ldots W_K & \text{or} \\ W_KW_{K-1}\ldots W_1 \end{cases}$$

$$P_2(K) = \begin{cases} W_2W_3\ldots W_{K+1} & \text{or} \\ W_{K+1}W_K\ldots W2 \end{cases}$$

Overlapping $p_1(K)$ and $p_2(K)$, there are 4 possible combinations:

$$\begin{cases} W_1W_2\ldots W_K \\ W_2W_3\ldots W_{K+1} \end{cases} \begin{cases} W_1W_2\ldots W_K \\ W_{K+1}W_K\ldots W_2 \end{cases} \begin{cases} W_KW_{K-1}\ldots W_1 \\ W_2W_3\ldots W_{K+1} \end{cases} \text{and} \begin{cases} W_KW_{K-1}\ldots W_1 \\ W_{K+1}W_K\ldots W_2 \end{cases}$$

which correspond to the 4 non-empty conditions in equation 14. No matter which combination occurs, we always have $p_1(K) \wedge p_2(K) = p(K+1)$.

$P_3 = \{p(3)\}$ are given by step 1 of the algorithm. By the description above, each p(4) can be split into $p_1(3)$, $p_2(3) \in P_3$. Moreover each pair of 3-paths in P_3 will be overlapped in step 2, therefore $P_4 = \{p(4)\}$ can be constructed by the algorithm. Go through the same process again until for some K, $P_{K+1} = \phi$. It means that all the nonchordal paths and circuits have length $\ell < K+1$. That is, we have obtained all the nonchordal paths, hence all the nonchordal circuits.

It follows that conclusion C1 and C2 are true.

$P_3 = \{p(3)\}$ does not include any duplications by step 1. Having obtained the jth 4-path (it may also be a 4-circuit), by the labelling algorithm, all the jth bits of the labels are set 1 for the $p(3)$'s each of which can be overlapped with some other $p(3)$ to produce the jth 4-path. For such $p(3)$'s, say, $p_1(3)$ and $p_2(3)$, $L(1,t) \cdot L(2,t) \neq 0$. By step 2, the operation $p_1(3) \wedge p_2(3)$ is not performed. Hence, there is no duplication in P_4. By induction, C3 is true.

If for some K, $P_K \neq \phi$ and $P_{K+1} = \phi$. It means that there is no nonchordal circuit containing $p(K)$, $p(K) \in P_K$. At the same time, if any $p(K)$ can be extended from an end vertex of it, then, the resulting path (circuit) must be a chordless path circuit. If not, $P_{K+1} \neq \phi$ is a contradiction. By definition 3 , C4 is true.

Q.E.D.

Now, we discuss another question. Suppose that all the nonchordal circuits of G are found, say, $C_1 \ldots C_m$. For a nonchordal n-circuit at least n-3 chords are required to make it into a chordal graph. If C_i, C_j have some common chords, then, the number of added edges may be less than $(n_i-3) + (n_j-3)$, where n_i, n_j are the lengths of C_i, C_j, respectively. However, in general, having added arbitrary $(n-3)$ or more chords to a nonchordal circuit C_i of length n, the resulting graph is not necessarily a chordal graph. In fact, we can show the following lemma (the proof is omitted).

Lemma 3.4. Assume that C is a nonchordal n-circuit. The chord di $(i=1,2,\ldots n-3)$ divides C into two circuits C'_{di} and C''_{di} with the common edge di. Let $d_1, \ldots d_{n-3}$ be $(n-3)$ added chords of C. Then C with their chords $d_1, \ldots d_{n-3}$ becomes a chordal graph if and only if for each d_i $(i=1,\ldots,n-3)$, d_j is a chord of either C'_{di} or C''_{di}, $\forall\, j \neq i$.

We introduce the following concepts and notations.

Let $\{C_1, C_2 \ldots\}$ be the set of nonchordal circuits, $D = \{d_1, \ldots d_K\}$ the set of common chords each member of which belongs to two circuits at least. Suppose that $m(d)$ is the multiplicity of $d \in D$. Let

$$D_i = \{d \mid d \text{ is a chord of } C_i \text{ and } d \in D\}$$

$I_i(d) = \{d' \mid d', d$ do not satisfy the condition in Lemma 2.2 and $d', d \in D_i\}$.

$I_i(d)$ is called the interference set of d in D_i.

$$g(d) = m(d) - 1$$

$g(d)$ is the number of saved chords whenever a set of chords containing d is added to G, and any pair of chords in the set are not included by the same interference set.

$$W(d) = m(d) - \sum_{d_r \in I_1(d) \cup \dots \cup I_K(d)} g(d_r)$$

$W(d)$ is an integer (not necessarily positive) and is called the weight of d.

Next, we establish the following algorithm.

Algorithm 3.3 (Chordal Graph Construction Algorithm)

Input The nonchordal circuits of G: $C_1, \dots C_m$.

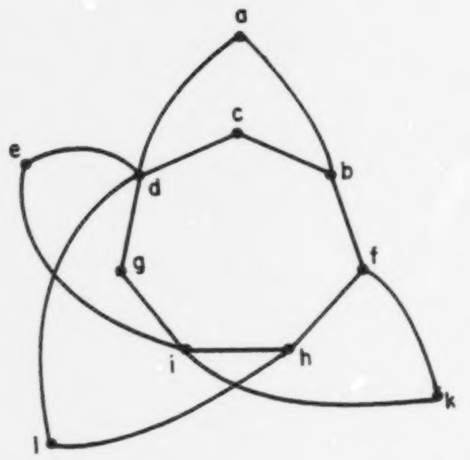
Output The added edges (chords) set $E"$.

Steps.
1. Find the common chordal set D and the multiplicities $m(d)$, $\forall\ d \in D_i$;
2. Find the common chordal subsets D_i contained by C_i, $\forall\ i$;
3. Find the interference set of d in D_i, $\forall\ d \in D$ and i;
4. Calculate the weight $w(d)$, $\forall\ d \in D$; $E" \leftarrow \phi$; $W \leftarrow \{w(d) | d \in D\}$;
5. Pick a chord d with the largest weight, ties are broken arbitrarily; Remove from W the maximum weight and the weights of the chords which belong to $I_1(d) \cup \dots \cup I_m(d)$; $E" \leftarrow E" \cup \{d\}$;
6. If $W \neq \phi$, go to 7; otherwise, go back to 5;
7. For the circuits which have not yet been transformed to chordal graphs, according to Lemma 3.4, add the chords to the circuits until they all become chordal graph. Suppose that such added edges are e_1, e_r, $E" \leftarrow E" \cup \{e_1, \dots e_r\}$, stop.

Example. The graph G in Fig. 24 has three nonchordal circuits 1326, 1325, 1526. There is only one common chord 12, we add 12 to G, a chordal graph with one added edge is obtained (see Fig. 25).

We discuss a more slightly complex example below.

Assume that we have been given 5 nonchordal circuits by Algorithm 3.2, they are shown in Fig. 31.



C_1	abcda
C_2	bfhigdcb
C_3	dgihld
C_4	abfkigda
C_5	bfkiedcb

Fig. 31

The common chord set D and the common chord subsets D_i $(i=1,\ldots 5)$ are listed in Table 3.

chord	contained by	m(d)	g(d)
bd	C_1,C_2,C_4,C_5	4	3
bi	C_2,C_4,C_5	3	2
bg	C_2,C_4	2	1
fi	C_2,C_4,C_5	3	2
fg	C_2,C_4	2	1
fd	C_2,C_4,C_5	3	2
fc	C_2,C_5	2	1
hg	C_2,C_3	2	1
hd	C_2,C_3	2	1
id	C_2,C_3,C_4,C_5	4	3
ic	C_2,C_5	2	1
bk	C_4,C_5	2	1
kd	C_4,C_5	2	1

Table 3

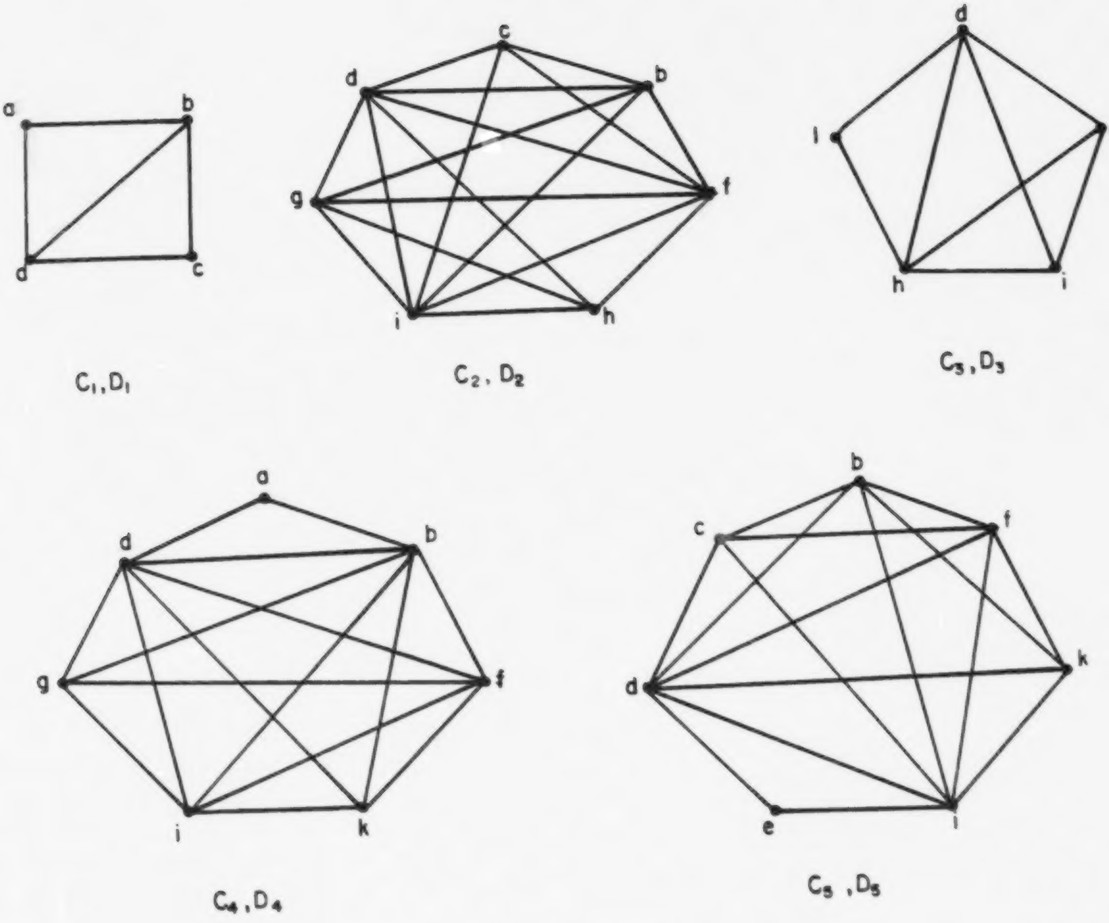
$D_1 = \{bd\}$
$D_2 = \{bd, bi, bg, fi, fg, fd, id, bk, dk, id, ic\}$
$D_3 = \{hg, hd, id\}$
$D_4 = \{bd, bi, bg, fi, fg, fd, id, bk, kd\}$
$D_5 = \{bd, bi, fi, fd, fc, id, ic, bk, kd\}$

The interference subsets and their unions $I(d) = I_1(d)U\ldots I_K(d)$ are listed in Table 4 (also see Fig. 32).

d	$I_1(d)$	$I_2(d)$	$I_3(d)$	$I_4(d)$	$I_5(d)$	$I(d)$
bd	ϕ	fc,ic	ϕ	ϕ	fc,ic	fc,ic
bi	ϕ	fc,fd,fg, hd,hg	ϕ	fd,fg,kd	fc,fd,kd	fc,fd,fg, hd,hg,kd
bg	ϕ	fc,fd,ic, id,hd	ϕ	fd,id,kd	ϕ	fc,fd,ic, id,hd,kd
fi	ϕ	hg,hd	ϕ	kb,kd	kb,kd	hg,hd,kb,kd
fg	ϕ	id,ic,bi, hd	ϕ	bi,kb,id, kd	ϕ	id,ic,bi, hd,kb,kd
fd	ϕ	ic,bg,bi	ϕ	kb,bi,bg	kb,bi,ic	ic,bg,bi,kb
fc	ϕ	bi,bg,bd	ϕ	ϕ	bd,bi,kb	bi,bg,bd,kb
hg	ϕ	fi,bi,ic, id	id	ϕ	ϕ	fi,bi,ic,id
hd	ϕ	fi,fg,bi, ic,bg	ϕ	ϕ	ϕ	fi,fg,bi, ic,bg
id	ϕ	hg,fg,bg	hg	bg,fg	ϕ	hg,fg,bg
ic	ϕ	bd,bg,fg, fd,hd,hg	ϕ	ϕ	kd,fd,bd	bd,bg,fg, fd,hd,hg,kd
bk	ϕ	ϕ	ϕ	fd,fg,fi	fc,fd,fi	fd,fg,fi,fc
kd	ϕ	ϕ	ϕ	fi,fg,bi,bg	fi,bi,ic	fi,fg,bi, bg,ic

Table 4

If we represent each nonchordal circuit as a convex polygon, the condition of Lemma 3.4 is equivalent to requiring that any two chords intersect only at the ends. The circuits and their D_i are shown in Fig. 32.



C_1, D_1 C_2, D_2 C_3, D_3

C_4, D_4 C_5, D_5

Fig. 32

The weights and their selection and elimination are given in Fig. 33. The process above produced 4 added edges. Having added the edges to G, the resulting graph still has one nonchordal circuit dihld; the edge il will be added. Finally, we obtain a chordal graph with the added edge set {bd,id,fi,fd,il}, as shown in Fig. 34.

Max W	d	W
●	bd	2
	bi	-4
	bg	-7
	fi	-1
	fg	-7
	fd	-2
	fc	-5
	hg	-6
	hd	-5
	id	1
	ic	-8
	bk	-4
	kd	-5

→

Max W	d	W
	bi	-4
	bg	-7
	fi	-1
	fg	-7
	fd	-2
	hg	-6
	hd	-5
●	id	1
	bk	-4
	kd	-5

→

Max W	d	W
	bi	-4
●	fi	-1
	fd	-2
	hd	-5
	bk	-4
	kd	-5

→

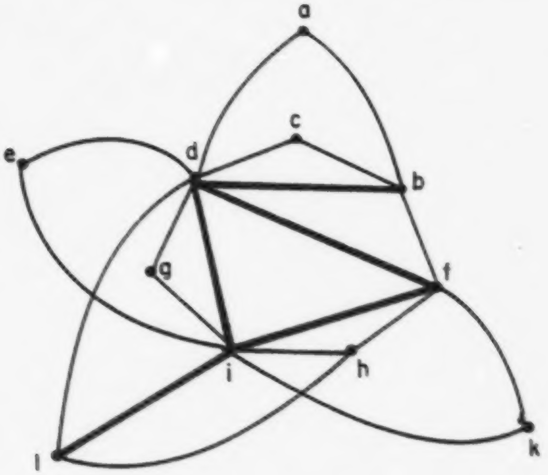
Max W	d	W
	bi	-4
●	fd	-2

↓

$\emptyset$

E" ← {bd,id,fi,fd}

Fig. 33



E" = {bd,fd,id,fi,il}

Fig. 34

Combining Algorithm 3.2 and 3.3, we can construct the chordal graph approximate model of the job compatibility graph G, if G is not a chordal graph. Because G is always a comparability graph, by the TRO Theorem (Appendix B), it does not contain nonchordal circuits of odd length. This fact can be used to check the programs for constructing graph G and implementing Algorithm 3.2.

Algorithm 3.2 and 3.3 constitute the first portion of the CCH method.

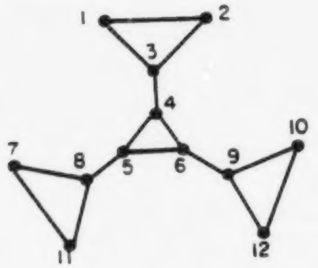
3.6 Minimum Covering of the Approximate Model by Cliques

We have established the chordal graph approximate model G' = (V,E') for the job compatibility graph G = (V,E), in the previous section. Our aim is to partition the job set J into subsets so that the jobs belonging to the same subset can be done simultaneously. There are two questions to be answered: (1) how many subsets can J be partitioned into, and (2) how to partition the set J into such subsets. In this and the next sections, we will discuss these questions in detail.

Every chordal graph G has a simplicial vertex, and it can be shown that the following statements are equivalent (see Appendix B).

(1) G is a chordal graph;
(2) G has a perfect vertex elimination scheme. Moreover, any simplicial vertex can start a perfect scheme.

A chordless graph can have a simplicial vertex. For a complete graph, each vertex is simplicial. If G is a chordal graph but not complete, it can have more than two simplicial vertices (see Fig. 35). For such a graph G, there exists at least one nonadjacent simplicial vertex for each simplicial vertex. This fact can be shown as follows. There are two nonadjacent vertices (say, a and b), since G is chordal graph and not complete. If c is the other simplicial vertex, then c is nonadjacent to either a or b. If it is not true, (i.e. a,b all are adjacent to c) then, a, b are adjacent, because c is a simplicial vertex.



simplicial vertices

1, 2, 7, 11, 10,12

Fig. 35

Given a chordal graph, an elimination scheme λ can be found in polynomial time (in fact, in linear time, [17]), and the algorithm is essentially a breadth-first search method. Applying the algorithm, $\lambda = (\lambda(1)\ldots\lambda(n))$ is tracked back. That is $\lambda(n)$ is found first, then, $\lambda(n-1)$ is found, until $\lambda(1)$ is found. We want to modify the method so that the previous vertex can be found more quickly. The method is very simple.

Algorithm (Elimination Scheme Algorithm)

1. Pick a simplicial vertex v arbitrarily;
2. Eliminate v and all the edges incident to it;
3. Repeat 1,2 on the remaining graph until only one vertex remains.

Obviously, the vertex sequence obtained by the algorithm gives an elimination scheme for a chordal graph.

We mentioned previously that the maximum independent set and minimum clique cover of a chordal graph can be found in polynomial time. The algorithm is given in [18]. Let G be a chordal graph and $\lambda = (\lambda(1)),\ldots\lambda(n))$ an elimination scheme. V(v) is the neighbour set of v. The algorithm is modified below so that it is suitable to our problem.

Algorithm (Maximum Independent Set and Minimum Clique Cover Algorithm)

1. $v_1 = \lambda(1)$;
2. v_i is the first vertex in λ which follow v_{i-1} and which is not in $V(v_1)\cup\ldots\cup(v_{i-1})$;
3. If there is not such vertex (at the point, $i=\alpha$), then $\{v_1\ldots v_\alpha\}$ is a maximum independent set and $v_1\cup V(V_1),\ldots v_\alpha\cup V(v_\alpha)$ gives a minimum clique cover.

It is easy to prove the conclusion in step 3 of the algorithm (see Appendix B).

Example. For graph G in Fig. 36, we have $\lambda = (abefcd)$ and

$v_1 = a$ $\qquad v_1\cup V(v_1) = \{a,b,e,f\}$
$v_2 = c$ $\qquad v_2\cup V(v_2) = \{c,d,b,f,e\}$

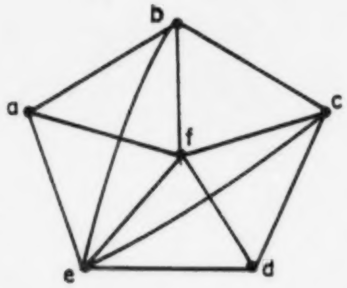
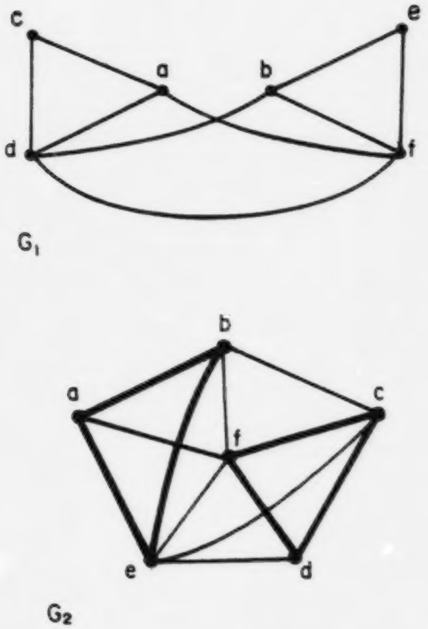


Fig. 36

Set $\{a,c\}$ is a maximum independent set and $c_1 = \{a,b,e,f\}$, $c_2 = \{c,d\} \subset v_2 \cup V(v_2)$ give a minimum clique cover $\{c_1,c_2\}$.

Every maximum independent set and every minimum clique cover are associated with an elimination scheme, in this algorithm. However, there are such maximum independent set and minimum clique cover which cannot be found by the algorithm. As an example, we consider the graph in Fig. 37.



Maximum independent sets

$\{a,e\}$ $\{d,e\}$
$\{b,c\}$ $\{c,f\}$
$\{a,b\}$

Minimum clique covers

$\{a,b,e,f\}$, $\{c,d\}$;
$\{c,d,e,f\}$, $\{a,b\}$;
$\{a,b,f\}$, $\{c,d,e\}$;
$\{a,b,e\}$, $\{c,d,f\}$

Fig. 37

For any elimination scheme λ, $\lambda(1)$ is a simplicial vertex, and any maximum independent set produced by the algorithm contains the vertex $\lambda(1)$. That is, the independent set contains at least a simplicial vertex. However, set $\{a,b\}$ contains no simplicial vertex. Similarly, a minimum clique cover produced by the algorithm contains at least one clique $v_i \cup V(v_i)$ where v_i is a simplicial vertex, however, the clique cover $\{\{a,b,f\},\{c,d,e\}\}$ contains no such clique. Therefore, $\{a,b\}$ and $\{\{a,b,f\},\{c,d,e\}\}$ cannot be found by the algorithm.

Now, we return to our approximate model $G' = (V,E')$. G' contains the added edges E''. Let $G = (V,E)$ be the job compatibility graph, then, it is evident that

$$\alpha(G) > \alpha(G') \tag{15}$$

and as a result of equation (15),

$$K(G) > K(G') \tag{16}$$

We know that $\alpha(G)$ can give us a lower bound for the makespan by equation (6).

$$\bar{M} = \max\left\{\left\lceil\frac{n}{m}\right\rceil, \alpha(G)\right\} \cdot p$$

We have discussed how to find $\alpha(G)$ in section 3.3. Obviously, finding $\alpha(G)$ is more difficult than finding $\alpha(G')$. In general, we cannot obtain $\alpha(G)$ from $\alpha(G')$ and G'. However, it is often possible to find an integer α such that $\alpha(G') < \alpha < \alpha(G)$, providing we make modifications to the elimination scheme algorithm and maximum independent set algorithm, and combine them.

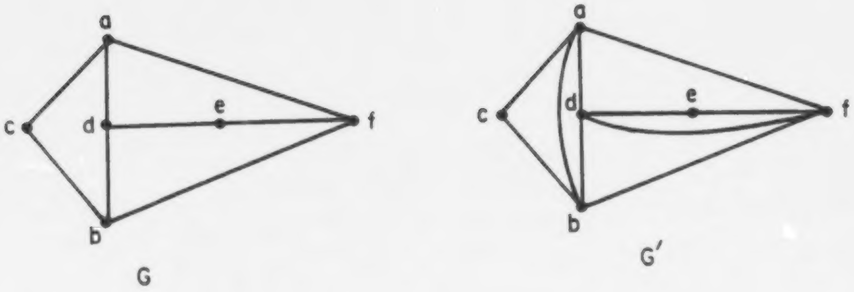
<u>Algorithm 3.4</u> (Finding an Independent Set of Size $\alpha > \alpha(G')$ for the job Compatiblity Graph G

Input approximate model $G' = (V,E')$, job compatibility graph $G = (V,E)$.

Output an independent set of size $\alpha > \alpha(G')$ of G.

Steps
```
1. $i \leftarrow 1$;
2. Find all the simplicial vertices of $G'$, say, $v_1 \dots v_k$;
3. For each simplicial vertex $v_j$ $(j=1,\dots K)$, count the $E''$-degree $r_j$ of $v_j$, i.e. the number of edges in $E''$ which are incident on $v_j$; pick arbitrarily $v$ having the minimum $r$, $\lambda(1) \leftarrow v$;
4. Pick $v \in V(v_1) \cup \dots \cup V(v_{i-1})$ having the minimum $r$, ties are broken arbitrarily. If there is no $v \in V(v_1) \cup \dots V(v_{i-1})$, pick $v$ having maximum $r$; $\lambda(i) \leftarrow v$;
   $G' \leftarrow G' - \{v\}$; $i \leftarrow i + 1$;
5. If $i < n$, go back to 2;
6. $u_1 \leftarrow \lambda(1)$; $i \leftarrow 2$;
7. Pick $u_i$, the first vertex in $\lambda$ which follows $u_{i-1}$ and which is not in $V_G(u_1) \cup \dots V_G(u_{i-1})$, where, $V_G(v)$ means the neighbour set of $v$ in graph $G$;
8. If there is no such vertex, then $\alpha \leftarrow i$ and $\{u_1, \dots u_\alpha\}$ is the desired independent set.
```

Example. We consider the graph in Fig. 38. The graph G' in Fig. 38 is its chordal graph approximate model.


a
c
d
e
f
b
a
c
d
e
f
b

G G'

Fig. 38

G' has two simplicial vertices c, e. $\lambda(1)$ can be either c or e. The evolution of the case $\lambda(1) = c$ is shown in Fig. 39.

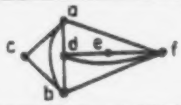
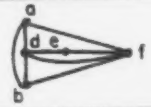
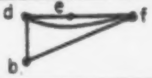
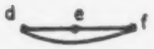
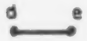
i	G'	Simplicial	r_i	$\lambda(i)$	Description
1		c e	$r_c = 0$ $r_e = 0$	c	$V(c) = \{a,b\}$
2		a b e	$r_a = 1$ $r_b = 1$ $r_e = 0$	a	$V(a) = \{b,d\}$
3		b e	$r_b = 0$ $r_e = 0$	b	$b \ \varepsilon \ V(c)\ V(a)$ $e \ \bar{\varepsilon} \ V(c)\ V(a)$
4		d e f	$r_d = 1$ $r_e = 0$ $r_f = 1$	f	$V(c)\ V(a)\ V(b) = \{a,b,d,f\}$
5		d e	$r_d = 0$ $r_e = 0$	e	$V(c)\ V(a)\ V(b)\ V(f) = \{a,b,d,f,e\}$
6		d	$r_d = 0$	d	λ = cabfed

Fig. 39

λ = cabfed
u_1 = c, $V_G(c) = \{a,b\}$
u_2 = f, $V_G(c) \cup V_G(f) = \{a,b,e\}$
u_3 = d
α = 3, $\{c,f,d\}$ is the desired independent set of G.

If $\lambda(1)$ was picked to be e, the evolution resembles Fig. 39. We have λ = efdacb. Then $\alpha = 3$, $\{e,a,b\}$ is the desired independent set. There are some alternatives for λ in both of the cases and we can always obtain the same α value, for the example. In fact, $\alpha = 3$ is the independence number of G, namely, $\alpha(G) = 3$.

There may be many clique covers for a given undirected graph G. Each clique induces a complete subgraph. Now, we consider the job compatibility graph $G = (V,E)$ and its chordal graph approximate model $G' = (V,E')$, and let $E'' = E' - E$. Assume that $\{C_1,\ldots C_K\}$ is a minimum clique cover of G' and $G'_1 = (C_1,E'_1),\ldots G'_K = (C_K, E'_K)$ the subgraph induced by $C_1,\ldots C_K$, respectively. The minimum clique covers of G' that interest us are those that minimize the size $|(E'_1 \cup \ldots \cup E'_K)\ E''|$. In other words, we hope that there are as few as possible added edges included in the subsets E'_i. We can modify the algorithms for the elimination scheme and for finding the minimum clique cover, then, combine these changed algorithms to form an algorithm for finding such minimum clique covers.

<u>Algorithm 3.5</u> (Finding a Minimum Clique Cover of G' with a Lower Value of $|E'_1 \cup \ldots \cup E'_K) \cap E''|$)

Input job compatibility graph $G = (V,E)$ and its chordal graph approximate model $G' = (V,E')$

Output minimum clique cover C of size $K = K(G')$.

Steps
```
1.  $i \leftarrow 1$;
2.  Find all the simplicial vertices of G', say, $v_1 \ldots v_K$;
3.  For each simplicial vertex v, (j=1,...K) count the number
    $d_j$ of the edges which are included by the complete subgraph
    $G'_j = (C_j,E'_j)$, where $C_j = v_j\ V(v_j)$, $G'_j$ is the induced
    subgraph of G' by $c_j$; determine the ratio $r_j = d_j/|E'_j|$;
    Pick arbitrarily v having the minimum r, $\lambda(1) \leftarrow v$;
4.  Pick $v \in V(v_1) \cup \ldots \cup V(v_{i-1})$ and v has the minimum r, ties
    are broken arbitrarily; if each simplicial vertex does not
    belong to $V(v_1) \cup \ldots \cup V(v_{i-1})$, pick the vertex v having the
    maximum r;
    $\lambda(i) \leftarrow v$; $G' \leftarrow G' - \{v\}$; $i \leftarrow i + 1$;
5.  If $i < n$, go back to 2;
6.  $u_1 \leftarrow \lambda(1)$; $i \leftarrow 2$;
7.  Pick $u_i$ as the first vertex in $\lambda$ which follows $u_{i-1}$ and
    which is not in $V(u_1) \cup \ldots V(u_{i-1})$, V(v) is the neighbour set
    of v in G';
8.  If there is no such vertex, then $K \leftarrow i$ and
    $\{u_1 \cup V(u_1),\ldots u_K \cup V(u_K)\}$ will give the desired minimum clique
    cover.
```

Example. The graph G' in Fig. 38 is not a good example for Algorithm 2.5, since G' has the unique minimum clique cover $\{\{c,a,b\},\{d,e,f\}\}$. All the added edges are included by the complete subgraphs. These facts will lead to some difficulties which will be addressed in the next section. Now, we consider another example (see Fig. 40).

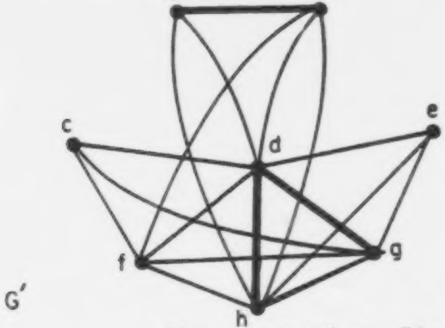


Fig. 40

The graph G' in Fig. 40 is a chordal graph approximate model of some G. The edges ab, gd, dh are added. The evolution of the example is shown in Fig. 41.

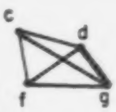
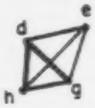
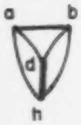
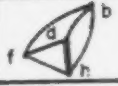
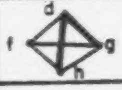
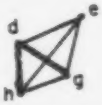
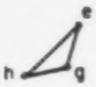
i	simplicial	C_j	$G_j = (C_j, E_j)$	r_j	$\lambda(i)$	description
1	c e a	c,d,g,f e,d,h,g a,b,d,h	  	$r_c = 1/6$ $r_e = 2/6$ $r_a = 1/6$	c	
2	e a	e,d,h,g a,b,d,h	id. id.	$r_e = 2/6$ $r_a = 1/6$	a	$r_a = \min\{r_e, r_a\}$
3	b e	b,d,f,h e,d,g,h	 id.	$r_b = 1/6$ $r_e = 2/6$	b	$b \in c\ V(c) \cup a \cup V(a)$
4	f e	f,d,g,h e,d,h,g	 id.	$r_f = 2/6$ $r_c = 2/6$	f	$f \in c \cup V(c) \cup a \cup V(a) \cup b \cup V(b)$
5	d e g h	d,e,g,h		$r = 2/6$	d	
6	e g h	e,g,h		$r = 0$	e	
7	g h	g,h		$r = 0$	g	
8	h	h		$r = 0$	h	λ = cabfdegh

Fig. 41

Thus, we have

$$\lambda = c\ a\ b\ f\ d\ e\ g\ h$$

and

$$\begin{aligned} u_1 &= c, & c \cup V(c) &= \{c,d,g,f\} \\ u_2 &= a, & a \cup V(a) &= \{a,b,d,h\} \\ u_3 &= e & e \cup V(e) &= \{e,d,h,g\} \end{aligned}$$

therefore

$$\begin{aligned} C_1 &= \{c,d,g,f\} \\ C_2 &= \{a,b,h\} \\ C_3 &= \{e\} \end{aligned}$$

the minimum clique cover is $\{C_1, C_2, C_3\}$.

3.7 Directed Hamiltonian Paths of a Tournament

For given chordal approximate model $G' = (G',E')$ of the job compatability graph, the minimum clique cover can be found. In this section, we will return to the compatability digraph and discuss the orientation of approximate model G' in order to partition the job set J into a group of subsets for each of which the jobs can be done simultaneously.

The chordal approximate model G' will be oriented in such a way that the job compatibility digraph D is a subgraph of the orientation of G'. The directions of the added edges are arbitrary. We call a complete directed graph a tournament. Let $D' = (V',A')$ be the orientation of G', $C = \{C_1,...C_K\}$ being the minimum clique cover of G' and $D'_1,...D'_K$ are the subgraphs of D' induced by $C_1,...C_K$, respectively. It is clear that $D'_i (i=1,...K)$ are tournaments. Assume that $D_1,...D_K$ are the subgraphs of D induced by $C_1,...C_K$ then $D_1,...D_K$ are acyclic since D is an acyclic digraph. If for some i, D'_i does not contain any added edge, then $D'_i = D_i$ and is an acyclic tournament.

The Hamiltonian directed path problem is solvable in polynomial time for both acyclic digraphs [9] and tournaments [19], [20]. For any i, D_i having a Hamiltonian directed path implies that the jobs constituting clique C_i can be done simultaneously. If there is no Hamiltonian directed path in D_i, we have to consider the tournament D'_i. If D'_i has a Hamiltonian directed path without any added edge and the path can be found, the same fact can be concluded. Otherwise, if a Hamiltonian directed path of D'_i with r added edges is found, the jobs constituting the clique C_i will have to be divided into $r-1$ groups (if the first and last arcs of the path are all added edges), or r groups (if either the first arc or last arc is added) or $r+1$ groups (if both the first and last arcs are not added).

We know that any tournament has a Hamiltonian directed path [21]. Further, a tournament may have more than one Hamiltonian directed path. In such a case we wish to obtain the Hamiltonian path with a lesser number of added edges. Note that we do not say "with the minimum number of added edges" here. To find a Hamiltonian directed path of D_i with the minimum number of added edges is equivalent to find the minimum weighted Hamiltonian directed path for a given tournament if we assign weight 1 to the original edges and weight 2 to the added edges. The latter problem may be NP-complete.

The algorithm to find a Hamiltonian directed path for an acyclic digraph can be found in [9]. The algorithm for solving the same problem for a tournament is given below.

<u>Algorithm A</u> (Hamiltonian Directed Path Generation in a Tournament)

Input a tournament D_n on n vertices with vertex set V and edge set E.

Output a Hamiltonian path $v_1 v_2 \cdots v_n$ of D_n.

Method
```
Begin $K \leftarrow 1$; $D \leftarrow D_n$; $A \leftarrow V$;
    Repeat choice such that $d_D^+(v_K) = \max_{v \in A} d_D^+(v)$, where $d_D^+(v)$ is the out-degree of v in D;
    $A \leftarrow \{v \in V \mid v_K v \in E\}$;
    $D \leftarrow D - v_K$, i.e., $V = V - \{v_K\}$ and $E = E - \{uv \mid u = v_K$ or $v = v_K\}$; $K \leftarrow K + 1$;
    Repeat until $K > n$
End
```

It has been shown that Algorithm A always generates a Hamiltonian path for given tournament D_n [19]. Now, two modifications will be made to the algorithm.

1. Algorithm A can only generate such a Hamiltonian path whose initial vertex has the maximum outdegree in tournament D_n. We can choose for v_1 a given vertex u when $K = 1$, and the $v_2, \ldots, v_n$ are chosen according to the original algorithm, then, a Hamiltonian path of initial vertex u will be found, if it exists. The modified algorithm is called Algorithm B.

2. If the indegree $d^-(v)$ is used instead of the outdegree $d^+(v)$ in Algorithm A, a Hamiltonian path will be found which has the maximum indegree vertex as its last vertex. If the modified algorithm is called Algorithm C, then Algorithm C always generates a Hamiltonian path. In reality, if D_n is a given tournament, then D_n' whose arcs have the opposite directions is a tournament and the performance of Algorithm C for D_n is equivalent to that Algorithm A for D_n'.

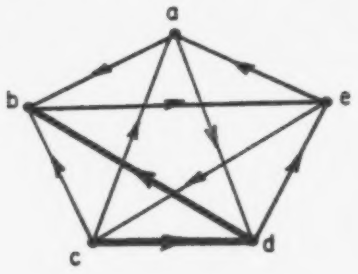
The modifications above can be combined to find a Hamiltonian path $u \rightarrow v_2 \rightarrow \ldots \rightarrow v_n$, where u is a given vertex. The algorithm finding the path (if it exists) is called Algorithm D.

For our problem, a Hamiltonian path with a lesser number of added edges is expected. The following algorithm will offer us more than one Hamiltonian path so that the choice of a path with a lesser number of added edges is possible.

Algorithm 3.6 (Hamiltonian Paths Generation in a Tournament)

Input	a tournament D_n on n vertices with vertex set V and edge set E.
Output	some Hamiltonian paths of D_n, one of them with the minimum number of added edges.
Steps	1. performing Algorithm B for all $v \in V$; 2. performing Algorithm D for all $v \in V$; 3. when a Hamiltonian path without added edges is found, the algorithm halts. Otherwise, pick the Hamiltonian path with the minimum number of added edges among all the resulting Hamiltonian paths.

Examples. A tournament of 5 vertices is shown in Fig. 42. The solution of Algorithm A is illustrated in Fig. 43.



cd, db are added edges

Fig. 42

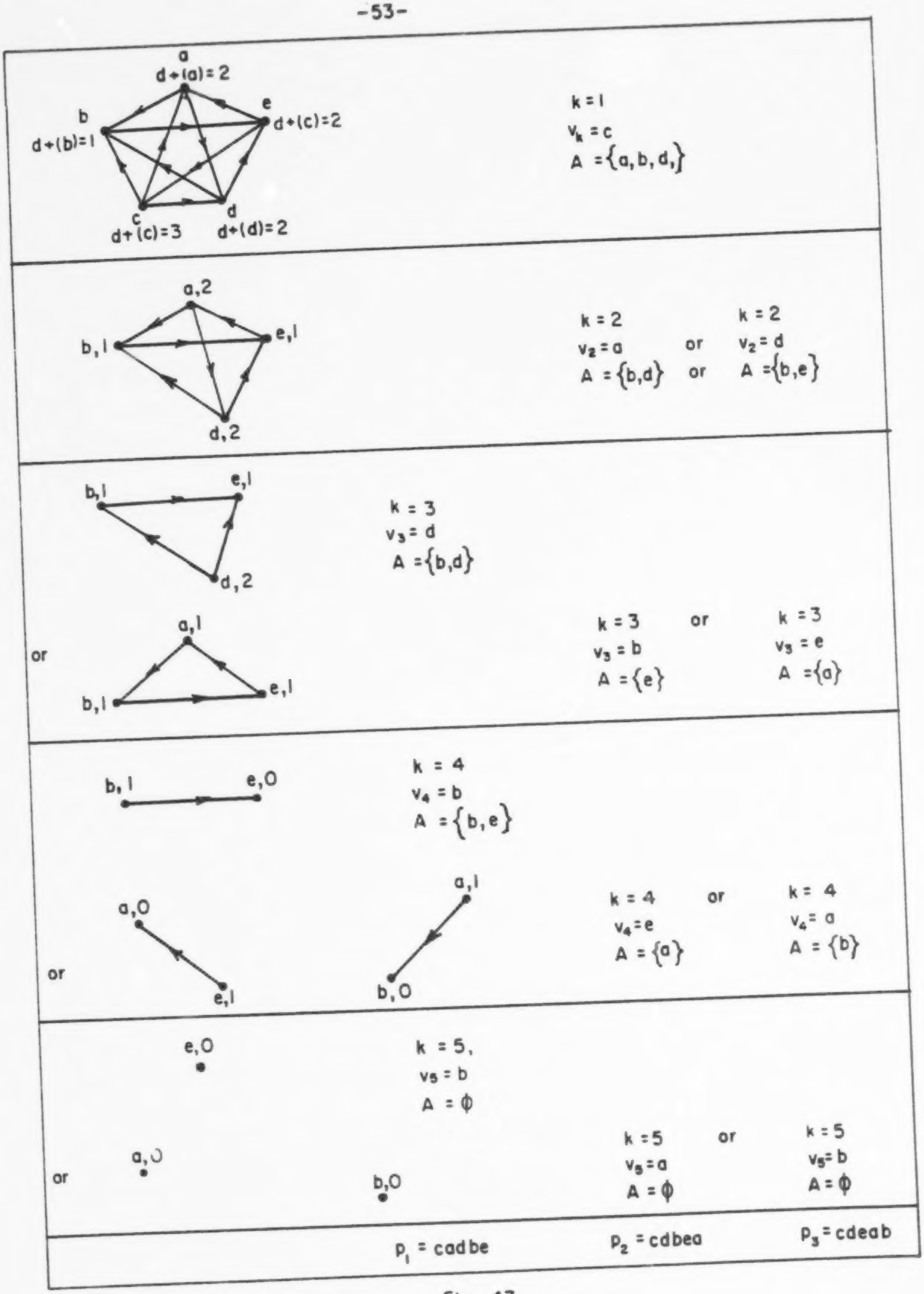

a
d+(a)=2
b
d+(b)=1
e
d+(c)=2
c
d+(c)=3
d
d+(d)=2
k=1
v_k=c
A={a,b,d,}
a,2
b,1
e,1
d,2
k=2
v_2=a
A={b,d}
or
or
k=2
v_2=d
A={b,e}
b,1
e,1
d,2
k=3
v_3=d
A={b,d}
or
a,1
b,1
e,1
k=3
v_3=b
A={e}
or
k=3
v_3=e
A={a}
b,1
e,0
k=4
v_4=b
A={b,e}
or
a,0
e,1
a,1
b,0
k=4
v_4=e
A={a}
or
k=4
v_4=a
A={b}
e,0
k=5,
v_5=b
A=ϕ
or
a,0
b,0
k=5
v_5=a
A=ϕ
or
k=5
v_5=b
A=ϕ
p_1=cadbe
p_2=cdbea
p_3=cdeab

Fig. 43

For the example, p_1 = cadbe having 1 added edge, p_2 = cdbea having 2 and p_3 = cdeab having 1. By observation, it is easy to see that there are 2 Hamiltonian paths without added edges: p_4 = adecb and p_5 = cbead. p_4 can be found by Algorithm B (u=a) but p_5 cannot. However, if we apply Algorithm C for u = d, p_5 can be found.

For a tournament on n vertices, $O(n^2)$ operations are required by Algorithm A [19]. Therefore Algorithm 3.6 requires $O(n^3)$ operations.

3.8 The CCH Method

To conclude this section, the main contents and steps of the CCH method are summarized as follows.

1. Generate an approximate model of the job compatibility graph, i.e., a chordal graph with a lesser number of added edges. To do this, we first have to recognize all the chordless circuits by Algorithm 3.2, then generate a chordal graph by adding edges with Algorithm 3.3.

2. Having obtained the chordal graph approximate model, it is necessary to partition the job set into a collection of subsets such that for each subset the jobs belonging to it can be done simultaneously. This task involves two steps. The first step is finding a minimum covering of the approximate model by cliques using Algorithm 3.5.

3. In the second step of the partitioning, the orientation of the approximate model will be considered. The subgraphs of the orientation induced by the cliques are tournaments. Algorithm 3.6 will be used to find some Hamiltonian paths for each tournament, and a Hamiltonian path with the minimum number of added edges will be selected from them.

4. If for some tournament, the resulting Hamiltonian path has r added edges, then it has to be divided into r-1, or r, or r+1 segments each of which does not include any added edges. The collection will consist of all the segments of the tournaments.

Any example which can effectively illustrate the entire method will be too complex, so we do not try to give such an example.

4.0 THE PCMI (PATH-CLIQUE-MATCHING-ITERATIVE) METHOD FOR GENERAL CRANE SCHEDULING PROBLEMS

4.1 State and State Compatibility

We have discussed the CCH method for the crane scheduling problems with equal job processing times. The CCH method, is a procedure for finding a schedule for which there is no delay due to crane interference between any two operations in a crane job. Now, we will consider another scheduling approach in which delay between crane operations is allowed. If the job processing times are not equal, such scheduling approaches are more significant, because the CCH method cannot be applied in such cases.

The general crane scheduling problem

$$u = \langle m,\ N,\ (n,n_j),\ r_j,\ \ell_{ji},\ p_{ji} \rangle$$

with assumptions (1)-(3) and the no passing constraint described in subsection 1.3 will be considered. Figure 44 shows a set of jobs and 3 different schedules.

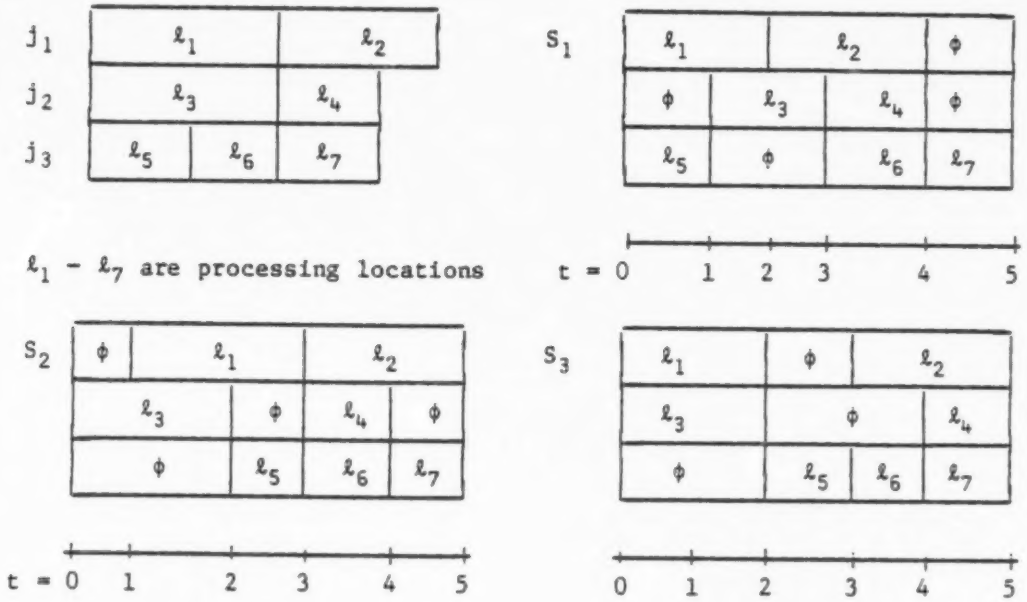


Fig. 44

The same job can be processed in different ways for the 3 schedules. Three different ways of processing job j_2 are shown in Fig. 45a. We can consider ϕ a special operation, called a virtual operation. A job containing virtual operations is called an extended job. Recall that we have introduced unitary job in subsection 3.2, and we can define similarly the unitary job for the extended job. For an extended job j of length ℓ, the corresponding unitary job is an ℓ-operation job of which every operation has the same processing time of 1 unit. The unitary jobs corresponding to the extended jobs in Fig. 45a are shown in Fig. 45b.

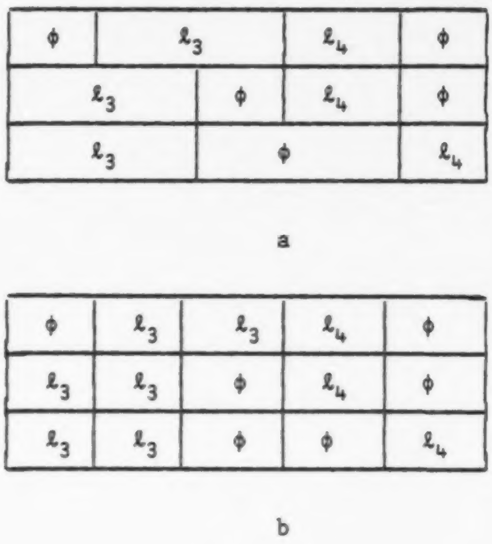

φ
ℓ_3
ℓ_4
φ
ℓ_3
φ
ℓ_4
φ
ℓ_3
φ
ℓ_4
a
φ
ℓ_3
ℓ_3
ℓ_4
φ
ℓ_3
ℓ_3
φ
ℓ_4
φ
ℓ_3
ℓ_3
φ
φ
ℓ_4
b

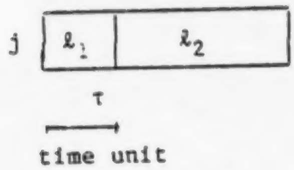
Fig. 45

Now, we introduce a concept called state, it is very important to our future discussion.

Definition 4.1 State

Let j be a job with processing time p_j and $\ell > p_j$ be a positive integer. j' is an extended job of j and having a length ℓ. S is the unitary job corresponding to j' and the processing location of each virtual operation of S can be defined arbitrarily. We call S a state of job j.

It is easy to see that for a given job, its state is a possible processing pattern. We note that for $\ell = p_j$, the extended job j' coincide with job j and hence has only one state, however for $\ell > p_j$, there are more than one state corresponding to the same job. The state set of job j is denoted by S(j). Figure 46 gives a job and its state set (the state length is 6).



S_1	ϕ	ϕ	ℓ_1	ℓ_2	ℓ_2	ℓ_2
S_2	ℓ_1	ϕ	ϕ	ℓ_2	ℓ_2	ℓ_2
S_3	ℓ_1	ℓ_2	ℓ_2	ℓ_2	ϕ	ϕ
S_4	ϕ	ℓ_1	ϕ	ℓ_2	ℓ_2	ℓ_2
S_5	ϕ	ℓ_1	ℓ_2	ℓ_2	ℓ_2	ϕ
S_6	ℓ_1	ϕ	ℓ_2	ℓ_2	ℓ_2	ϕ

$S(j) = \{S_1, S_2, S_3, S_4, S_5, S_6\}$

Fig. 46

Suppose that J is the job set, $S = \{S(j) | j \in J|\}$ being the state set of J. The following two binary relations on S and the graphs describing them can be defined.

Definition 4.2 Relation $\overset{s}{\sim}$ and State Compatibility Graph

Let s_r, $s_t \in S$ be two states of length ℓ, and $s_r \in S(j_1)$, $s_t \in S(j_2)$, j_1, $j_2 \in J$. We say that $s_r \overset{s}{\sim} s_t$ if the following conditions are satisfied:

(1) $j_1 \neq j_2$;
(2) ℓ_{ri}, ℓ_{ti} are the ith operations of s_r, s_t $(1 \leq i \leq \ell)$ respectively and ℓ_{ri}, $\ell_{ti} \neq \phi$;
(3) for all ℓ_{ri}, ℓ_{ti} satisfying condition (2), inequality

$$\ell_{ri} < \ell_{ti} \tag{17}$$

holds, or for all the ℓ_{ri}, ℓ_{ti}, inequality

$$\ell_{ri} > \ell_{ti} \tag{18}$$

holds. Obviously, $\overset{s}{\sim}$ (simply, $\sim$) is a binary relation on state set S.

If for states s_r, s_t we have $\ell_{ri} < \ell_{ti}$, by definition 4.2, $s_r \sim s_t$. On the other hand, inequalities $\ell_{ti} > \ell_{ri}$ imply that $s_t \sim s_r$ from definition 4.2. Therefore, binary relation $\sim$ is a symmetric relation; and can be represented by an undirected graph $G' = (V', E')$ called state compatibility graph, for which each vertex represents a state and $u, v \in V'$, $uv \in E'$ if and only if $u \overset{s}{\sim} v$.

Obviously, for s_r and s_t, if $\ell = p_{j_1} = p_{j_2}$, the relations $s_T \overset{s}{\sim} s_t$ and $j_1 \overset{j}{\sim} j_2$ are the same.

Definition 4.3 Relation $\overset{s}{\rightarrow}$ and State Compatibility Digraph

Let s_r, $s_t \in S$ be two states of length ℓ, we say that $s_r \overset{s}{\rightarrow} s_t$, if the following conditions are satisfied:

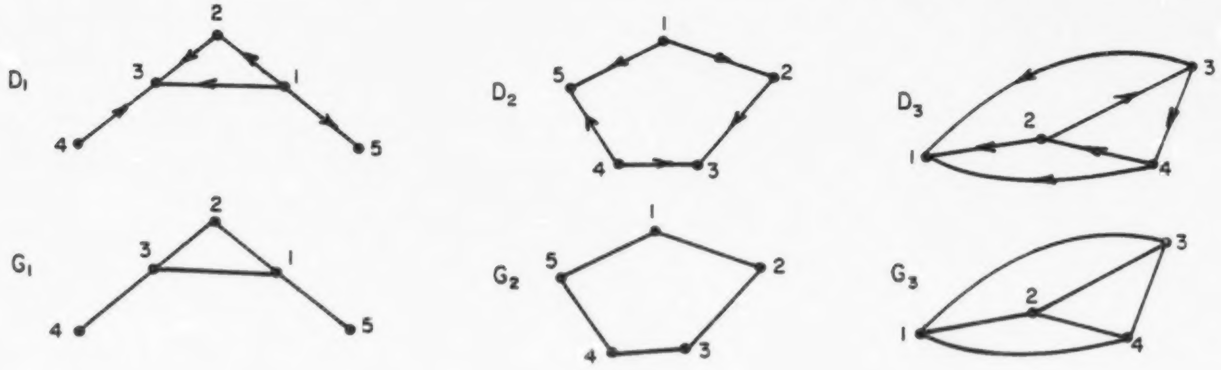
(1), (2) are the same conditions (1), (2) in Definition 4.2;
(3) $\ell_{ri} < \ell_{ti}$ for all ℓ_{ri}, $\ell_{ti} \neq \phi$ $(1<i<\ell)$.

Relation $\overset{s}{\rightarrow}$ is also a binary relation on state set S. It can be represented by a digraph D' = (V',A') called a state compatibility digraph in which each vertex represents a state and u, v $\in$ V', (u,v) $\in$ A' if and only if $u \overset{s}{\rightarrow} v$.

Obviously, if $s_r \rightarrow s_t$, then, $s_r \sim s_t$. Conversely, if $s_r \underset{s}{\sim} s_t$, we have either $s_r \rightarrow s_t$ or $s_t \rightarrow s_r$. If $\ell = p_{j_1} = p_{j_2}$ relations $s_r \overset{s}{\rightarrow} s_t$ and $j_1 \overset{j}{\rightarrow} j_2$ are the same.

For convenience, the prime ' of G' or D' will be removed and the state compatibility graph and digraph will be denoted by G = (V,E) and D = (V,A) respectively.

In general, $|S|$ is larger than $|J|$, therefore G and D have many vertices and edges (arcs). Some subsets of S and subgraphs of G and D are shown in Fig. 47.



S_1	4	φ	2	2	2
S_2	φ	6	5	5	5
S_3	5	7	7	7	φ
S_4	2	2	6	6	φ
S_5	7	7	φ	3	3

S_1	φ	7	18	18	φ
S_2	9	9	φ	φ	8
S_3	11	11	φ	14	14
S_4	6	6	19	φ	10
S_5	φ	8	20	20	12

S_1	6	6	φ	9	φ
S_2	3	φ	8	8	φ
S_3	4	4	φ	φ	7
S_4	φ	5	6	6	φ

Fig. 47

In Fig. 47, D_1 is a transitive orientation but D_2, D_3 not; D_1, D_2 are acyclic but D_3 not. G_1, G_3 are comparability graph (G_2 is not) and D_1 is a transitive orientation of G_1, but D_3 is not a transitive orientation of G3. These examples show that relations $\overset{s}{\to}$, D' and G' are different from $\overset{j}{\to}$, D and G. Relation $\overset{j}{\to}$ is transitive, hence G is a comparability graph and D is a transitive orientation of G, moreover D is acyclic. However, relation $\overset{s}{\to}$ is not transitive, therefore G' and D' are a general graph and digraph, respectively.

Generally, the state set for each job has more than one state. Suppose that j_1 and j_2 are given jobs and for states $s_1 \in S(j_1)$, $s'_1 \in S(j_2)$, we have $s_1 \twoheadrightarrow s'_1$. Note that it is possible that there exists $s'_2 \in S(j_2)$ such that $s_1 \to s'_2$ (or $s'_2 \to s_1$), or that there exists $s_i \in S(j_1)$, $s'_K \in S(j_2)$ such that $s_i \to s'_K$ and so on (see Fig. 48). This fact will lead us to find those states rather than jobs which can be done simultaneously.

j_1	10	9	15
j_2	14	5	12

s_1	φ	10	9	15
s_2	10	φ	9	15
s_3	10	9	φ	15
s_4	10	9	15	φ

s'_1	φ	14	5	12
s'_2	14	φ	5	12
s'_3	14	5	φ	12
s'_4	14	5	12	φ

$s_1 \twoheadrightarrow s'_1$
$s_2 \twoheadrightarrow s'_2$
$s_3 \twoheadrightarrow s'_3$
$s_4 \twoheadrightarrow s'_4$

$s'_2 \to s_1$
$s_2 \to s'_4$
$s'_3 \to s_1$

Fig. 48

In general, each state contains one or more virtual operations. Therefore, if we obtained a group of states which can be done simultaneously, then, a subschedule with idleness can be constructed. In this section, state and state compatibility will be discussed instead of job and job compatability. Only states with the same length will be considered. For given job compatibility G = (V,E) and job compatiblity approximate model G' = (V',E'), the ratio $\beta = (|E'|-|E|)/|E|$ has been used as a performance measure of G'. Because relation $\overset{s}{\to}$ is not transitive, there may not be an arc (v_1, v_3) if $v_1 \to v_2$ and $v_2 \to v_3$ in the state compatibility digraph. Therefore, in general, more added edges are needed to obtain a chordal graph approximate model for state compatibility graph, that is β may be too large. Hence it is not very good to extend the CCH method. Another method called the PCMI (Path-Clique-Matching-Iterative) will be discussed in this section.

4.2 N-Matrix

It is possible that the jobs associated with the states included by the same directed path in the state compatibility digraph cannot be done simultaneously because the relation $\overset{s}{\to}$ is not transitive. In order to partition the job set J into subsets for which every job can be assigned a state and the resulting states belonging to the same subset can be done simultaneously, such vertex-disjoint directed paths must be found for which the relation $\overset{s}{\to}$ is transitive, i.e., if s_1, s_2, s_3 belong to the same path and $s_1 \to s_2$, $s_2 \to s_3$, then $s_1 \to s_3$.

Hence two problems arise: 1. How to find such directed paths (they will be called T-directed paths); 2. Having found the paths, how to select those for which any two paths are vertex-disjoint. In order to establish the algorithms to solve these problems, we will introduce a specific matrix called the N-matrix.

The two problems can be illustrated in the following example. Assume that:

$m = 2$
j_1: 1,6
j_2: 2,4; $p_1 = p_2 = 2$.

Let the state length be 3, then the state sets and state compatibility digraph are shown in Fig. 49.

$v_1 = S_{11}$	φ	1	6
$v_2 = S_{12}$	1	φ	6
v3 = S_{13}	1	6	φ

(states for j_1)

$v_4 = S_{21}$	φ	2	4
$v_5 = S_{22}$	2	φ	4
$v_6 = S_{23}$	2	4	φ

(states for j_2)

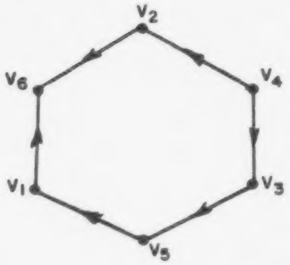
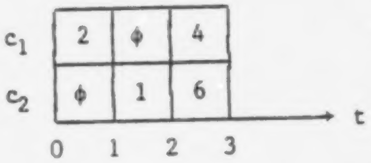


Fig. 49

The T-directed 2-paths: v_1v_6, v_5v_1, v_3v_5, v_4v_3, v_4v_2, v_4v_6, v_2v_6.
The state disjoint 2-paths: v_5v_1, v_2v_6, v_4v_3.
The job disjoint 2-path: v_5v_1 (or v_2v_6, or v_4v_3).
The schedule is shown in Fig. 50.



makespan = 3 and this is also optimal

Fig. 50

Definition 4.4 N-Matrix

$S = \{1,2,\ldots n\}$ is the set of the first n natural numbers, we call a nxn matrix $N = (n_{ij})$ an N-matrix, if $n_{ij} = 0$ or being the following sequence ($\forall\ i_j \in S$)

$$((S_1, m_1), \ldots (S_{r(ij)}, m_{r(ij)})) \; ; \tag{19}$$

where
1) $r(ij)$ is any positive integer;
2) (S_i, m_i), $i = 1, \ldots r$ are 2-tuples and for all i, $S_i \quad S$, $m_i \in S$
3) $|S_1| = \ldots = |S_r|$ for any n_{ij}.

Examples. The following matrices are N-matrices of order 3 (for simplicity, the parentheses of the 2-tuples are omitted), for convenience sequence $((S_1, m_1), \ldots (S_r, m_r))$ is written as

$$\begin{matrix} S_1, m_1 \\ \cdot \\ \cdot \\ \cdot \\ S_r, m_r \end{matrix}$$

$$N_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \!\!\! \begin{array}{c} \begin{array}{ccc} 1 \qquad & \qquad 2 \qquad & \qquad 3 \end{array} \\ \left(\begin{array}{ccc} \begin{matrix} \{2\},1 \\ \{3\},2 \end{matrix} & 0 & \{2\},3 \\ 0 & \begin{matrix} \{3\},2 \\ \{1\},2 \end{matrix} & \begin{matrix} \{1,2\},3 \\ \{2,3\},1 \end{matrix} \\ \{3\},1 & \{3\},2 & \begin{matrix} \{2\},1 \\ \{3\},1 \end{matrix} \end{array} \right) \end{array}$$

$n_{22} = ((\{3\},2),(\{1\},2))$
$n_{23} = ((\{1,2\},3),(\{2,3\},1))$

$$N_2 = \begin{pmatrix} \begin{matrix}\{2\},1 \\ \{3\},1\end{matrix} & 0 & 0 \\ \begin{matrix}\{1,2\},2 \\ \{2,3\},1 \\ \{1,3\},3\end{matrix} & \begin{matrix}\{1\},2 \\ \{3\},2\end{matrix} & \begin{matrix}\{1\},1 \\ \{3\},2\end{matrix} \\ 0 & 0 & \begin{matrix}\{1\},3 \\ \{2\},3\end{matrix} \end{pmatrix} \qquad n_{12} = ((\{1,2\},2),(\{2,3\},1), (\{1,3\},3),\dots\dots\dots$$

Definition 4.5 The Addition of the Elements of N-Matrices

The addition of the elements follows the following rules.

The sum of two zero-elements is a zero-element, namely, $0 + 0 = 0$;
The sum of a zero-element and a nonzero element is the same nonzero element, namely, $0 + n_{ij} = n_{ij}$ $(n_{ij} \neq 0)$;
The sum of two nonzero elements $x = ((S_1,m_1),\dots(S_r,m_r))$ and $y = ((S'_1,m'_1),\dots(S'_\ell,m'_\ell))$ is the nonzero element Z

$$z = x + y = ((S_1,m_1),\dots(S_r,m_r),\ (S'_1,m'_1),\dots(S'_\ell,m'_\ell)) \tag{20}$$

Definition 4.6 The Multiplication of the 2-Tuples

Assume that (S_i,m_i), (S'_i,M'_i) are two 2-tuples. Then

$$(S_i,m_i)(S'_i,m'_i) = \begin{cases} (S_i \cup \{m_i\},\ m'_i) & \text{if } S_i \subseteq S_i \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

Definition 4.7 The Multiplication of the Elements of N-Matrices

Assume that $x, y \neq 0$ and

$$x = ((S_1,m_1),\dots(S_r,m_r))$$
$$y = ((S'_1,m'_1),\dots(S'_\ell,m'_\ell)).$$

Then,

$$\begin{aligned} z &= xy \\ &= ((S_1,m_1)(S'_1,m'_1),\dots(S_1,m_1)(S'_\ell,m'_\ell),\dots(S_r,m_r)(S'_1,m'_1),\dots \\ &\quad (S_r,m_r)(S'_\ell,m'_\ell)). \end{aligned} \tag{22}$$

If $x = 0$ or $y = 0$, then $z = xy = 0$.

Definition 4.8 Multiplication of N-Matrices

Assume that $A = (a_{ij})$ and $B = (b_{ij})$ are nxn N-matrices, the product $C = (c_{ij})$ of A and B is obtained as the multiplication of normal matrices. That is,

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}.$$

Obviously, the product of two nxn N-matrices is also a nxn N-matrix.

Suppose that G = (V,E) is an undirected graph: $|V| = n$, $V = \{v_1, \ldots v_n\}$. The degree of vertex v_j is denoted by dj ($j=1,\ldots n$). Let $v_{j_1}, \ldots v_{jdj}$ be the vertices which are adjacent to v_j (the neighbors of v_j) and $U_j = \{j_1, \ldots j_{dj}\}$. For simplicity, notations $\{v_{j_1}, \ldots . v_j d_j\}$ and U_j will sometimes be considered equivalent.

Definition 4.9 Matrices A(G) and B(G)

Matrices $A(G) = (a_{ij})$ and $B(G) = (b_{ij})$ are defined as follows:

$$a_{ij} = \begin{cases} (\{i\},j) & \text{if } i < j \text{ and } v_i v_j \in E; \\ 0 & \text{the conditions above do not hold simultaneously.} \end{cases} \tag{23}$$

$$b_{ij} = \begin{cases} (U_j,j) & \text{if } i > j \text{ and } v_i v_j \in E: \\ 0 & \text{the conditions above do not hold simultaneiously} \end{cases} \tag{24}$$

Obviously, A(G) and B(G) are N-matrices and are upper triangular matrix and lower triangular matrix with zero-diagonal elements, respectively.

Example. For the graph in Fig. 51, matrices A(G) and B(G) are as follows:

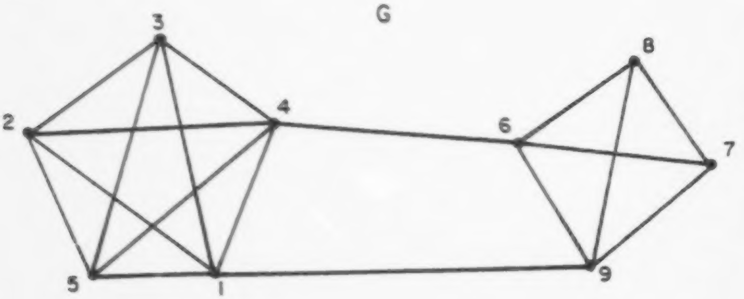


Fig. 51

$$A(G) = \left(\begin{array}{ccccccccc|c}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \\
 & \{1\},2 & \{1\},3 & \{1\},4 & \{1\},5 & & & & \{1\},9 & 1 \\
 & & \{2\},3 & \{2\},4 & \{2\},5 & & & & & 2 \\
 & & & \{3\},4 & \{3\},5 & & & & & 3 \\
 & & & & \{4\},5 & \{4\},6 & & & & 4 \\
 & & & & & & & & & 5 \\
 & & & & & & \{6\},7 & \{6\},8 & \{6\},9 & 6 \\
 & & & & & & & \{7\},8 & \{7\},9 & 7 \\
 & & & & & & & & \{8\},9 & 8 \\
 & & & & & & & & & 9
\end{array}\right)$$

$$B(G) = \left(\begin{array}{c|ccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
1 & & & & & & & & & \\
2 & \{2,3,4,5,9\},1 & & & & & & & & \\
3 & \{2,3,4,5,9\},1 & \{1,3,4,5\},2 & & & & & & & \\
4 & \{2,3,4,5,9\},1 & \{1,3,4,5\},3 & \{1,2,4,5\},3 & & & & & & \\
5 & \{2,3,4,5,9\},1 & \{1,3,4,5\},2 & \{1,2,4,5\},3 & \{1,2,3,5,6\},4 & & & & & \\
6 & & & & \{1,2,3,5,6\},5 & & & & & \\
7 & & & & & & \{4,7,8,9\},6 & & & \\
8 & & & & & & \{4,7,8,9\},6 & \{6,8,9\},7 & & \\
9 & \{2,3,4,5,9\},1 & & & & & \{4,7,8,9\},6 & \{6,8,9\},7 & \{6,7,9\},8 &
\end{array}\right)$$

Suppose that $\Gamma = (V,A)$ is a directed graph. The in-degree of vertex v_j is denoted by d_j^- which is the number of arcs directed to v_j and $v_{j_1},\ldots v_{j_{d_j^-}}$ are the vertices from which the arc originate. The set of positive integers $j_1,\ldots j_{d_j^-}$ is denoted by U_j^-, namely $U_j^- = \{j_1,\ldots j_{d_j^-}\}$. Sometimes we will consider the notations $\{v_{j_1},\ldots v_{j_{d_j^-}}\}$ and U_j^-, or v_j and j to be equivalent.

Definition 4.10 Matrices A(D) and $B^-(D)$

The matrices $A(D) = (a_{ij})$ and $B^-(D) = (b^-_{ij})$ are defined as follows:

$$a_{ij} = \begin{cases} (\{i\},j) & (v_i,v_j) \in A \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

$$b_{ij} = \begin{cases} (U_j^-, j) & (v_i, v_j) \in A \\ 0 & \text{otherwise} \end{cases} \qquad (26)$$

Obviously A(D) and B⁻(D) are also N-matrices.

Example. For the directed graph D in Fig. 52, the matrices A(D) and B⁻(D) are as follows:

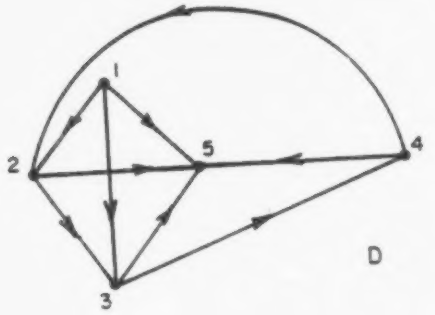


Fig. 52

$$A(D) = \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left(\begin{array}{ccccc} & \{1\},2 & \{1\},3 & & \{1\},5 \\ & & \{2\},3 & & \{2\},5 \\ & & & \{3\},4 & \{3\},5 \\ & \{4\},2 & & & \{4\},5 \\ & & & & \end{array}\right) \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \qquad (27)$$

$$B^-(D) = \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left(\begin{array}{ccccc} & \{1,4\},2 & \{1,2\},3 & & \{1,2,3,4\},5 \\ & & \{1,2\},3 & & \{1,2,3,4\},5 \\ & & & \{3\},4 & \{1,2,3,4\},5 \\ & \{1,4\},2 & & & \{1,2,3,4\},5 \\ & & & & \end{array}\right) \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \qquad (28)$$

Definition 4.11 Matrix $A^{(K)}(G)$

Suppose that $A(G) = (a_{ij})$, $B(G) = (b_{ij})$ are the matrices in Definition 4.9, then, matrix $A^{(K)}(G) = (a_{ij}^{(K)})$, $K > 2$ is defined recurrently as follows:

$$a_{ij}^{(2)} = a_{ij}$$

$$a_{ij}^{(K)} = \begin{array}{ll} \text{the } (i,j)\text{th element of matrix } A^{(K-1)}(G)\ B(G) & K > 2,\ i < j \\ 0 & K > 2,\ i > j \end{array} \tag{29}$$

Definition 4.12 Matrix $A^{(K)}(D)$

Suppose that $A(D) = (a_{ij})$, $B^{-}(D) = (b^{-}_{ij})$ are the matrices in Definition 4.10, then, matrix $A^{(K)}(D) = (a_{ij}^{(K)})$, $K > 2$ is defined as follows:

$$A^{(2)}(D) = A(D),$$

$$A^{(K)}(D) = A^{(K-1)}(D)\ B^{-}(D) \tag{30}$$

Obviously, $A^{(K)}(G)$ and $A^{(K)}(D)$ are N-matrices, and $A^{(K)}(G)$ is an upper triangular matrix with zero-diagonal elements, but $A^{(K)}(G)$ (or $A^{(K)}(D)$) is not the Kth power of $A(G)$ (or $A(D)$).

Examples. For the graph G in Fig. 51 we have (the blanks express zero-elements)

$$A^{(3)}(G)$$

$$= \begin{cases} \text{the } (i,j)\text{th element of } A^{(2)}(G)\ B(G) & i < j \\ 0 & i > j \end{cases}$$

$$= \begin{cases} \text{the } (i,j)\text{th element of } A(G)\ B(G) & i > j \\ 0 & i < j \end{cases}$$

$$= \begin{array}{c|ccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline 1 & & \begin{array}{l}\{1,3\},2\\ \{1,4\},2\\ \{1,5\},2\end{array} & \begin{array}{l}\{1,4\},3\\ \{1,5\},3\end{array} & \{1,5\},4 & & & & & \\ 2 & & & \begin{array}{l}\{2,4\},3\\ \{2,5\},3\end{array} & \{2,5\},4 & & & & & \\ 3 & & & & \{3,5\},4 & & & & & \\ 4 & & & & & & & & & \\ 5 & & & & & & & & & \\ 6 & & & & & & & \begin{array}{l}\{6,8\},7\\ \{6,9\},7\end{array} & \{6,9\},8 & \\ 7 & & & & & & & & \{7,9\},8 & \\ 8 & & & & & & & & & \\ 9 & & & & & & & & & \end{array} \tag{31}$$

For the diagraph D in Fig. 52, we have

$$A^{(3)}(D)$$

$$= A^{(2)}(D)\ B^{-}(D)$$

$$= A(D)\ B(D)$$

$$= \begin{array}{cccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & & & \{1,2\},3 & & \begin{matrix}\{1,2\},5\\ \{1,3\},5\\ \{2,3\},5\\ \{3,4\},5\\ \{4,2\},5\end{matrix} \\ 2 & & & & & \\ 3 & & & & & \\ 4 & & & & & \\ 5 & & & & & \end{array} \qquad (32)$$

4.3 The Algorithm for Finding T-Directed K-Path and K-Cliques

Definition 4.13 Transitive Directed Path

Assume that $D = (V,A)$ is a digraph and $p = v_1 \ldots v_K$ is a directed path of D, if $(v_i,v_j) \in A$ for any v_i, $v_j \in p$, p is called a transitive directed path, simply a T-directed path.

A path with K vertices will be called a K-path. For a given digraph $D = (V,A)$ and $K < |V|$, an algorithm finding all T-directed K-paths of D will be described in which the matrices A(D) and $B^{-}(D)$ will be used.

Algorithm 4.1 The T-Directed K-Path Algorithm

Input digraph $D = (V,A)$, positive interger $K < |V|$

Output matrix $A^{(K)}(D)$ whose non-zero elements give all T-directed K-path of D.

Steps
1. Construct matrices A(D) and $B^{-}(D)$ according to Definition 4.10;
2. $i \leftarrow 3$, $A^{(2)}(D) = A(D)$;
3. If $i < K$, calculate $A^{(i)}(D)$ according to recurrence formula (30), i.e. $A^{(i)}(D) = A^{(i-1)}(D)\ B^{-}(D)$;
4. If $A^{(i)}$ is a zero-matrix, then there is no T-directed K-path; otherwise, if $i < K$, $i \leftarrow i+1$, go to 3; if $i = K$, stop.

Example. Assume that D is the digraph in Fig. 52, the matrices A(D), $B^{-}(D)$ and $A^{(3)}(D)$ were given in equations 27, 28 and 32.

From $A^{(3)}(D)$ we know that there are 6 T-directed 3-paths, 123, 125, 135, 235, 345, 425. If we want to find all T-directed 4-paths, matrix $A^{(4)}(D)$ must be calculated. In fact, $A^{(4)}(D)$ has only one non-zero element: $a_{15}^{(4)} = (\{1,2,3\},5)$. It implies that there is only one T-directed 4-path: 1235. We know that there is no T-directed 5-path because $A^{(5)}(D)$ is a zero-matrix.

Now, suppose that $D = (V,A)$ is the state compatibility digraph rather than a normal digraph. Then p being a T-directed path of D implies that the relation § is transitive for the states constituting p. That is, the states can be done simultaneously.

The state compatibility digraph D is different from a normal digraph. Since a job may have more than one state, the vertices corresponding to the same job j constitute subset $V(j) = S(j)$ of V and all S(j) satisfy:

$$S(j_1) \cap S(j_2) = \phi \quad , \quad \text{for any } j_1 \neq j_2 \tag{33}$$

$$(u,v) \bar{\in} A, \quad \text{if } u, v \in S(j) \tag{34}$$

Hence, if p is a T-directed path, then any two vertices of p do not belong to the same S(j), because a job cannot be assigned to two cranes. Note that if p is a normal directed path in the state compatibility digraph D, it is possible that two vertices not successive in p may belong to the same S(j) (see Fig. 53 the heavy lines). If p is a T-directed path, the dotted lines are required. However $(u,v) \bar{\in} A$ because u,v belong to the same collection $S(j_1)$. Therefore, if there are two vertices in p belonging to the same collection, then p cannot be a T-directed path.

Thus, Algorithm 4.1 can be used for finding the T-directed paths in the state compatibility digraph.

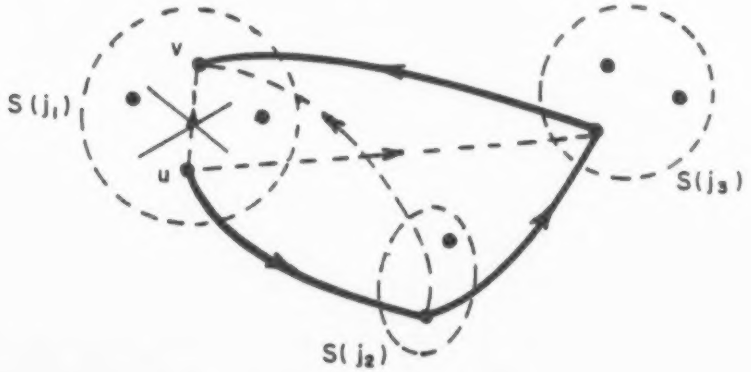


Fig. 53

An algorithm for finding all K-cliques of a given undirected graph G can be obtained if a few changes are made to Algorithm 4.1.

Algorithm 4.2 The K-Clique Algorithm

Input undirected graph $G = (V,E)$, positive integer $K \leq |V|$.

Output maxtrix $A^{(K)}(G)$ whose non-zero elements give all K-cliques of G.

Steps.

1. Construct matrices $A(G)$ and $B(G)$ according to definition 4.9;
2. $i \leftarrow 3$, $A^{(2)}(G) = A(G)$;
3. If $i \leq K$, calculating $A^{(i)}(G)$ according to recurrence formula (29)

$$a_{sr}^{(i)}(G) = \begin{cases} \text{the } (s,r)\text{th element of } A^{(i-1)}(G)\, B(G) & s < r \\ 0 & s \geq r \end{cases};$$

4. If $A^{(i)}$ is a zero-matrix, then there is no K-clique; otherwise, if $i < K$, $i \leftarrow i+1$, go to 3; if $i = K$, stop.

Example. The matrix $A^{(3)}(G)$ (equation (31)) has been calculated for graph G in Fig. 44. $A^{(3)}(G)$ shows that there are 14 3-cliques in G: $\{1,3,2\}$, $\{1,4,2\}$, $\{1,5,2\}$, ... $\{6,9,7\}$, $\{6,9,8\}$, $\{7,9,8\}$. The matrix $A^{(4)}(G)$ can be calculated as follows:

$$a_{s,r}^{(4)}(G) = \begin{matrix} \text{the } (s,r)\text{th element of } A^{(3)}(G)\, B(G) & s < r \\ 0 & s \geq r \end{matrix}$$

$$A^{(4)}(G) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 7 & 9 \\ \hline 1 & & \begin{matrix}\{1,4,3\},2 \\ \{1,5,3\},2 \\ \{1,5,4\},2\end{matrix} & \{1,5,4\},3 & & \\ 2 & & & \{2,5,4\},3 & & \\ \cdot & & & & & \\ \cdot & & & & & \\ \cdot & & & & & \\ 6 & & & & \{6,9,8\},7 & \\ \cdot & & & & & \\ \cdot & & & & & \\ 9 & & & & & \end{array} \qquad (35)$$

It follows that G has only 6 4-cliques: $\{1,4,3,2\}$, $\{1,5,3,2\}$, $\{1,5,4,2\}$, $\{1,5,4,3\}$, $\{2,5,4,3\}$, $\{6,9,8,7\}$. We can calculate $A^{(5)}(G)$:

$$A^{(5)}(G) = \begin{array}{c} \\ 1 \\ \vdots \\ 9 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \cdots & 9 \end{array} \\ \left(\begin{array}{cccc} & \{1,5,4,3\},2 & & \\ & & & \\ & & & \\ & & & \end{array} \right) \end{array} \tag{36}$$

It follows that G has only one 5-clique $\{1,5,4,3,2\}$ which is also the maximum clique of G.

Up to now, we have not shown that Algorithms 4.1 and 4.2 can give all the T-directed K-paths and K-cliques, respectively. We will now give the proof for Algorithm 4.2.

Assume that $G = (V,E)$ is a simple undirected graph. $|V| = n$ and all the vertices are numbered as $1,2,\ldots,n$, i.e., $V = \{1,2,\ldots n\}$. We have the following lemmas.

Lemma 4.1 Assume that (S,m) is a 2-tuple in matrix $A^{(K)}(G)$, then, the following statements are true:

1. $S \subset V$, $m \in V$, and $|S| = K-1$;
2. Let $S = \{j_1, j_2, \ldots, j_{K-1}\}$, $m = j_K$, and $j_1 = \min_{j \in S} j$, then (S,m) belongs to the (j_1, j_K)th element of $A^{(K)}(G)$ and $j_1, j_2, \ldots, j_K$ satisfies the following inequalities:

$$j_2 > j_3 > \ldots > j_K > j_1 \tag{37}$$

Proof. 1. By the definitions of $A(G)$, $B(G)$, $A^{(K)}(G)$ and the multiplication of N-matrices, the statement is true obviously.

2. If $K = 2$, the proof is trivial because G is a simple graph. Assume that the statement is true for $K = t$. Let $(\{j_1, j_2, \ldots j_t\}, j_{t+1})$ be a 2-tuple of $A^{(t+1)}(G)$, then there exists 2-tuple $(\{j_1, j_2, \ldots, j_{t-1}\}, j_t)$ in $A^{(t)}(G)$ and 2-tuple $(U_{j_{t+1}}, j_{t+1})$ in $B(G)$ such that $(\{j_1, j_2, \ldots, j_t\}, j_{t+1}) = (\{j_1, j_2, \ldots j_{t-1}\}, j_t)\,(U_{j_{t+1}}, j_{t+1})$. Since $(\{j_1, j_2, \ldots j_{t-1}\}, j_t)$, denoted by (S, j_t), belongs to the (j_1, j_t)th element of $A^{(t)}(G)$ and $(U_{j_{t+1}}, j_{t+1})$ being the (j_t, j_{t+1})th element of $B(G)$, so $(\{j_1, j_2, \ldots j_t\}, j_{t+1})$, denoted by (S', j_{t+1}), belongs to the (j_1, j_{t+1})th element of $A^{(K+1)}(G)$. By induction, we have $j_1 = \min_{j \in S} j$. Since $A^{(t)}(G)$

is an upper triangular maxtrix, so $j_1 < j_t$. Therefore $j_1 = \min_{j \in S'} j$.

By induction, we know that

$$j_2 > j_3 > \dots > j_t > j_1 .$$

Since B(G) is a lower triangular matrix and $A^{(K+1)}(G)$ being an upper triangular matrix, so $j_{t+1} < j_t$ and $j_1 < j_{t+1}$. It follows that

$$j_2 > j_3 > \dots > j_t > j_{t+1} > j_1 .$$

Q.E.D.

Lemma 4.2 Two-Tuple (S,m) belongs to an element of $A^{(K)}(G)$ if and only if $S \quad \{m\}$ is completely connected.

Proof. 1. If (S,m) belongs to an element of $A^{(K)}(G)$, then $S \cup \{m\}$ is completely connected. To show this, the inductive procedure is applied. For K = 2, the statement is valid obviously. Assume that for any 2-tuple (S_t, m_t) of $A^{(t)}(G)$, set $S_t \cup \{m_t\}$ is completely connected when K = t. Let (S_{t+1}, m_{t+1}) be any 2-tuple of $A^{(t+1)}(G)$, denoted by (S', m'). Suppose that $(S', m') \in a^{(t+1)}_{i,m'}$, where $a^{(t+1)}_{i,m'}$ is the (i,m')th element of $A^{(t+1)}(G)$, then there must be a 2-tuple $(S^{(t)}, m^{(t)})$, denoted by (S,m), $(S,m) \in a^{(t)}_{im}$ and a 2-tuple $(U_{m'}, m') = b_{mm'}$ such that $(S',m') = (S,m)(U_m, m') = (S_t \{m\}, m')$ where $a^{(t)}_{im}$ and $b_{mm'}$ are the (i,m)th element of $A^{(t)}(G)$ and the (m,m')th element of B(G), respectively. By induction, $S \cup \{m\}$ is completely connected, since $b_{m,m'} \neq 0$ vertices m, m' are adjacent. Moreover $S \subseteq U_{m'}$ being the neighbours of vertex m'. Hence the set $S' \quad \{m'\} = S \cup \{m\} \cup \{m'\}$ is completely connected.

2. If $S \quad \{m\}$ is completely connected where $S = \{j_1, j_2, \dots j_{K-1}\} \subseteq V$ and $m = j_K \in V$, then (S,m) is a 2-tuple in $A^{(K)}(G)$.

The induction for K is applied again. For all K, assume that $j_2, j_3, \dots j_K, j_1$ are ranked in decreasing order, i.e. $j_2 > j_3 > \dots > j_K > j_1$. Then, for K = 2, the proof is trivial. Suppose that for K = t the statement is valid and $\{j_1, j_2, \dots, j_t, j_{t+1}\}$ being a set completely connected, and $j_2 > j_3 > \dots > j_{t+1} > j_1$. Then $\{j_1, j_2, \dots, j_t\}$ is also completely connected. By induction, $(\{j_1, j_2, \dots j_{t-1}\}, j_t)$ is a 2-tuple in $A^{(t)}(G)$. Since j_t, j_{t+1} are adjacent and $j_t > j_{t+1}$, so $b_{jt,jt+1} \neq 0$ and $b_{jt,jt+1} = (U_{jt+1}, j_{t+1})$. Consider the product $(\{j_1, j_2, \dots, j_{t-1}\}, j_t)(U_{jt+1}, j_{t+1})$. Obviously $\{j_1, j_2, \dots, j_{t-1}) \subset U_{jt+1}$, because $U_{jt+1} \supseteq \{j_1, j_2, \dots, j_{t-1}, j_t\}$. Hence, $(\{j_1, j_2, \dots, j_{t-1}\}, j_t)(U_{jt+1}, j_{t+1}) = (\{j_1, j_2, \dots j_t\}, j_{t+1})$ is a 2-tuple in $A^{(t+1)}(G)$.

Q.E.D.

Now, the following theorem can be shown.

Theorem 4.3 Assume that $A^{(K)}(G)$ is the matrix resulting from Algorithm 4.2 for a given undirected graph G. Then, the 2-tuples contained by $A^{(K)}(G)$ correspond one-to-one to the K-cliques of G. In particular, if for some $i < K$, $A^{(i)}(G) = 0$, then there is no clique of size K in G.

Proof. We know that from Lemma 4.2 any 2-tuple of $A^{(K)}(G)$ corresponds to a K-clique of G and vice versa. We only need to show that this is a one-for-one correpondance. In fact, if C is a K-clique of G, the vertices in C can be ranked in decreasing order and let $\{j_1, j_2, \ldots, j_K\}$ be the ranked K-clique, for which $j_2 > j_3 > \ldots > j_K > j_1$. If there are two 2-tuples corresponding to C, say, $(S_1, m_1) = (\{j_{11}, \ldots\}, j_{K1})$ and $(S_2, m_2) = (\{j_{12}, \ldots.\}, j_{K2})$, by Lemma 4.1, $j_{11} = j_{12} = j_1$, $j_{K1} = j_{K2} = j_K$. Thus $S_1 = S_2$ and $m_1 = m_2$, i.e. $(S_1, m_1) = (S_2, m_2)$. Conversely, if $(S, m) = (\{j_1, j_2, \ldots, j_{K-1}\}, j_K)$ is a 2-tuple of $A^{(K)}(G)$, then $j_1, j_2, \ldots, j_K$ satisfying $j_2 > j_3 > \ldots > j_K > j_1$ determine uniquely a set $\{j_1, j_2, \ldots j_K\}$ which is completely connected and $j_{i1} \neq j_{i2}$, if $i1 \neq i2$. That is, $\{j_1, j_2, \ldots j_K\}$ is the unique K-clique of G corresponding to (S,m).

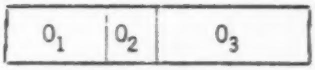
Q.E.D.

It is evident that the same K-clique of G occurs once and only once both in $A^{(K)}(G)$ and in all intermediate resulting $A^{(i)}(G)$. Note that expressing a K-clique as a vertex sequence, a K-clique can correspond to K! different sequences. Therefore to find all the K-cliques of G by an algorithm, the same K-clique may be enumerated K! times. It is true that the storage requirement and number of operations will be cut down greatly if Algorithm 4.2 is applied to find all K-cliques of G.

If $K = |V|$, Algorithms 4.1 (with some changes) and 4.2 can be used to find the directed Hamiltonian path and the maximum clique, respectively. We know that the two problems are NP-complete for a general digraph or an undirected graph. Therefore, one cannot expect that Algorithm 4.1 or 4.2 can solve the problems in polynomial time. However, if K is a constant not related to $|V|$, the algorithms can find all T-directed K-path and K-cliques respectively in polynomial time. The algorithms will be combined into the PCMI method.

4.4 State Tree

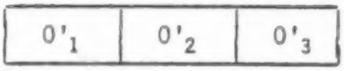
We mentioned in section 4.1 that there can be several states belonging to the same job. The number of the states depends on the length ℓ of the states, the number n_j of operations contained by the job j, and the job processing time p_j. Two jobs with the same n_j and p_j, and their state sets with the same state length $\ell = p_j + 1$ are shown in Fig. 54.



job j_1

ϕ	O_1	O_1	O_2	O_3	O_3	O_3
O_1	O_1	ϕ	O_2	O_3	O_3	O_3
O_1	O_1	O_2	ϕ	O_3	O_3	O_3
O_1	O_1	O_2	O_3	O_3	O_3	ϕ

The state set $S(j_1)$



job j_2

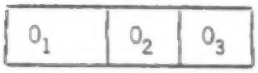
time unit

ϕ	O'_1	O'_1	O'_2	O'_2	O'_3	O'_3
O'_1	O'_1	ϕ	O'_2	O'_2	O'_3	O'_3
O'_1	O'_1	O'_2	O'_2	ϕ	O'_3	O'_3
O'_1	O'_1	O'_2	O'_2	O'_3	O'_3	ϕ

The state set $S(j_2)$

Fig. 54

We can say that $S(j_1)$ and $S(j_2)$ have the same structuring. The meaning of "the same structuring" is: 1. the same size; 2. the same relative position between the valid operations and virtual operations. The states belonging to the same job can be given uniquely as long as the operation processing locations and times are specified, if the structuring of the state set is known. Note that, some state sets can have the same structuring even though the state lengths are different for different state sets. For example, for the state set $S(i)$ in Fig. 55 the state length is 5, it is different from the length 7 of $S(j_1)$ and $S(j_2)$ in Fig. 54, however their state sets have the same structuring.



job i

time unit

ϕ	O_1	O_1	O_2	O_3
O_1	O_1	ϕ	O_2	O_3
O_1	O_1	O_2	ϕ	O_3
O_1	O_1	O_2	O_3	ϕ

The state set $S(i)$

Fig. 55

Clearly, two state sets belonging to different jobs j_1, j_2 respectively have the same structuring if and only if $n_{j_1} = n_{j_2}$ and $\ell_{j_1} - p_{j_1} = \ell_{j_2} - p_{j_2}$, where ℓ_{j_1}, ℓ_{j_2} are the state lengths for $S(j_1)$ and $S(j_2)$ respectively.

Definition 4.14 State tree

Assume that j is a job, S(j) is a state set belonging to j. A tree is called a state tree of j and denoted by T(S(j)) if its pendant vertices correspond one-to-one to the states of S(j).

If S(j) and S(i) are state sets with the same structuring (i,j may be the same job or may not), it is evident that the states of S(j) correspond one-to-one to the states of S(i). Therefore, the state sets with the same structuring have the same state tree. A procedure for producing the state tree for a given state set structuring is necessary for our further discussion.

Algorithm 4.3 Constructing the State Tree

Input: Positive integers n_j and $r = \ell - p_j + 1$, where n_j is the number of operations, ℓ is the state length and p_j is the job processing time.

Output: A labelled directed tree. For each vertex v, a label L(v) (an r-bit Boolean vector) is assigned.

Steps: All vertices v_{ij} with the same value of i (j=1,2,...) will be lined up on the same horizontal line and any arc (u,v) pointing downward.

1. Initializing: $i = 0$, $j = 1$, v_{ij} is the root of the tree and $L(V_{ij}) = 0$ (a 0-vector with r components);
2. For $i = 1$, $j = 1, \ldots r$, v_{ij} are the successors of the root and $L(v_{ij}) = a_j$, where a_j are the r-bit Boolean vectors with a 1 in position j and 0 in all other positions;
3. For $1 < i < n_j$, if $L(v_{ij}) = a_K$, then v_{ij} has $r - K + 1$ succesors $v_{i+1,j_1}, \ldots v_{i+1,j_{r-K+1}}$ with the labels a_K, $a_{K+1}, \ldots a_r$, respectively.

Example. A job (n_j=3) and its state tree (with the state length p_j+2) are shown in Fig. 36.

j	0_1	0_2	0_3

S_1	ϕ	ϕ	0_1	0_1	0_2	0_3	0_3
S_2	0_1	0_1	ϕ	ϕ	0_2	0_3	0_3
S_3	0_1	0_1	0_2	ϕ	ϕ	0_3	0_3
S_4	0_1	0_1	0_2	0_3	0_3	ϕ	ϕ
S_5	ϕ	0_1	0_1	ϕ	0_2	0_3	0_3
S_6	ϕ	0_1	0_1	0_2	ϕ	0_3	0_3
S_7	ϕ	0_1	0_1	0_2	0_3	0_3	ϕ
S_8	0_1	0_1	ϕ	0_2	ϕ	0_3	0_3
S_9	0_1	0_1	ϕ	0_2	0_3	0_3	ϕ
S_{10}	0_1	0_1	0_2	ϕ	0_3	0_3	ϕ
	t_o	t_o+1	t_o+2				t

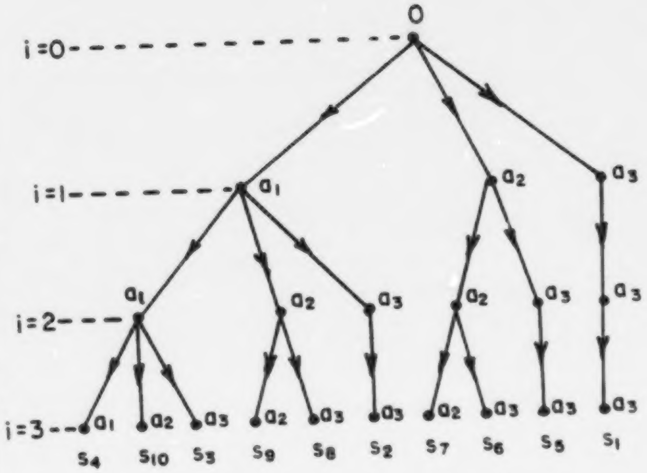


Fig. 56

It must be shown that this algorithm generates precisely a state tree for a given job and state length. Assume that job $j = (0_1, 0_2, \ldots 0_{n_j})$ with job processing time p_j is given and the state length being considered is ℓ. Then, n_j and $r = \ell - p_j + 1$ determines a labelled directed tree by Algorithm 4.3.

For the states of length ℓ of job j, the start time of any operation of j can only occur in a time interval of length r units, each state having exactly (r-1) virtual operations, $0'_1, 0'_2, \ldots 0'_{r-1}$. Obviously, the start time interval for operation 0_i is $I_i = [a, a+\ell-p_j]$ where $a = 1 + \sum_{K=1}^{i-1} p_{jK}$ and p_{jK} being the processing time of operation 0_K. For any given state, the start time (precisely, the relative start time) t_i of each operation 0_i $(i=1,2,\ldots n_j)$ is uniquely determined satisfying $(a+t_i-1) \in I_i$ where t_i is a positive integer. An r-bit Boolean vector a_i will be assigned to the start time t_i, the t_ith component of a_i being 1 and the others being 0. In other words, the position of each virtual operation in the state is also determined.

The following inequalities must be satisfied:

$$t_i < t_{i+1}\ , \qquad i = 1,\ldots n_j-1 \tag{38}$$

That is, an operation cannot be started before its previous operation is finished.

For the states shown in Fig. 56, we have $I_1 = [1,3]$, $I_2 = [3,5]$ and $I_3 = [4,6]$, and for S_1, $t_1 = 3$, $t_2 = 3$, $t_3 = 3$; for S_3 $t_1 = 1$, $t_2 = 1$, $t_3 = 3$, and so on.

Conversely, given r-bit Boolean vectors a_i $(i=1,2,\ldots n_j)$ of which the ith component is 1 and the others being 0, and t_i $(i=1,2,\ldots n_j)$ satisfying (38), we can uniquely determine a state of length $\ell = r + p_j - 1$. Hence, one-to-one correspondance has been established between the states and all possible vector groups $\{a_i | i=1,2,\ldots,n_j-1,\ t_i < t_{i+1}\}$'s for given job and state length ℓ.

The vector groups $\{a_i | i=1,2,\ldots n_j-1,\ t_i<t_{i+1}\}$'s correspond one-for-one to the paths of the state tree T from the root to each pendant vertex. To show this, let $u_1 u_2 \ldots u_{n_j}$ be such a directed path whose length must be n_j, and $L(u_1) = a_{t_1}, \ldots, L(u_{n_j}) = a_{tn_j}$. By step 3, we have

$$t_i < t_{i+1},\quad i = 1,\ldots n_j-1$$

That is, a vector group $\{a_{ti} | i=1,2,\ldots,n_j,\ t_i<t_{i+1}\}$ is uniquely determined. Conversely, let $\{a_i | i=1,\ldots n_j\}$ be a group of the r-bit Boolean vectors for which only the ith component is 1 and $t_i (i=1,\ldots n_j)$ satisfies (38), than we can pick the vertex u_1 at level 1 with label a_{t_1}, the vertex u_2 from the successors of u_1 with label a_{t_2}, until the vertex u_{n_j} with label $a_{t_{nj}}$ is picked, so we can find a directed path of length n_j corresponding to it. Obviously, the path of length n_j in T must have an end of pendant vertex.

Thus, the following theorem has been shown.

Theorem 4.4 The labelled directed tree produced by Algorithm 4.3 for given positive integers n_j and r is a state tree of the jobs with n_j operations, the difference between the state length and job processing time being the constant (r-1).

For given n_j and r, the state tree can be denoted by $T(n_j,r)$. Note again that for jobs with different job processing times and for states with different state lengths, the state trees can be the same, as long as the jobs have equal numbers of operations and $(\ell-p)$ is a constant, where ℓ is the state length and p is the job processing time.

4.5 Algorithms for Examining the Suitability of Jobs to Blanks

Consider some stage of the process of assigning jobs to cranes. Assume that J is the set of unassigned jobs and S is the subschedule. A crane may be idle for a time interval T in the subschedule S. In other words, if a subschedule is represented by a Gantt chart as shown in Fig. 2, then there exist some blanks (delay periods denoted by ϕ) in the subschedule. Each blank corresponds to a crane c and a time interval T. It is possible that there exists a job $j \in J$ which can be done in T by crane c. A match of the maximum size between the unassigned jobs and blanks is required.

Let b be a blank in S. The length of the corresponding time interval T is also called the length of b and denoted by ℓ. For given blank b, two vectors of ℓ-dimension will be specified and denoted by X, Y respectively. Some examples are shown below to illustrate how to specify X and Y.

Figure 57 gives a schedule. Any schedule (or subschedule) can be regarded as an array in which each cell is denoted by S_{ij} if it is in the ith row and jth column. Consider 4 blanks, b_1, b_2, b_3 and b_4 of the schedule (there are also other blanks denoted by ϕ).

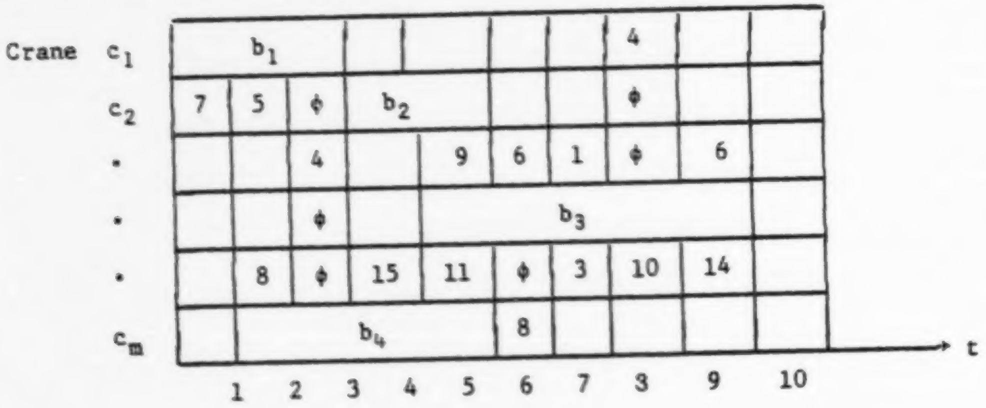


Fig. 57

$$X_1 = (0,0,0) \quad , \quad Y_1 = (7,5,4)$$

$$X_3 = (9,6,1,4,6), \quad Y_3 = (11,8,3,10,14)$$

$$X_4 = (8,4,15,11) \ , \quad Y_4 = (\infty,\infty,\infty,\infty,)$$

Vectors X and Y will be used to determine which unassigned jobs and which states of the jobs are suitable for filling the blanks.

Assume that B is the blank set of subschedule S. $b \in B$ is a blank of length ℓ_b. For each unassigned job $j \in J$ with length $p_j, < \ell_b$, a pair of integers (n_j, r_{bj}) can be determined, where n_j is the number of operations of j and $r_{bj} = \ell_b - p_j + 1$. Let R', R be the following sets:

$$R' = \{(n_j, r_{bj}) \mid \forall\, j \in J \text{ and } b \in B \text{ satisfying } p < \ell_b, \text{ where } p \text{ is the minimum job length for } J\} \qquad (40)$$

$$R = \{(n_j, r_{bj}) \mid (n_j, r_{bj}) \in R' \text{ and } (n_j, r_{bj}) \neq (n_{j'}, r_{b'j'}), \text{ if } j \neq j', b \neq b\} \qquad (41)$$

In addition, for each pair of positive integers $(n,r) \in R$, a set $Q(n,r)$ can be determined by:

$$Q(n,r) = \{(j,b) \mid n_j = n,\ r_{bj} = r,\ j \in J,\ b \in B\} \qquad (42)$$

and let Q be the collection of $Q(n,r)$:

$$Q = \{Q(n,r,) \mid (n,r) \in R\} \qquad (43)$$

For a given (n,r) the corresponding state tree T(n,r) is unique, and vice versa. So, if $T = \{T(n,r) \mid (n,r) \in R\}$ is the state tree set, then there is a one-to-one correspondance between Q and T. For given (n,r) if $|Q(n,r)| > 1$, there are more than one (j,b) corresponding to T(n,r), say $Q(n,r) = \{(j_1, b_1), (j_2, b_2) \ldots, (j_K, b_K)\}$. It is possible that blank b appears more than once in a given Q(n,r). Assume that B(n,r) is the set of distinct blanks appearing in Q(n,r). For $b \in B(n,r)$, let $b_1 = b_2 = \ldots = b_i = b$, and $J_b(n,r) = \{j_1, j_2, \ldots, j_i\}$ for which $(j,b) \in Q(n,r)$, $j = j_1, j_2, \ldots, j_i$.

The vertices of a state tree are positioned on n_j levels by Algorithm 4.3. There is only one vertex on the level zero, it being the root of the tree, r vertices on level 2, $\sum_{K=1}^{r} \frac{K(1+K)}{2}$ vertices on level 3, and so on. Every vertex expresses a subtree, in other words, a subset of the state set. For the state tree in Fig. 56, the left most vertex with label a_1 on level 2 expresses such a subset of which each state starts the first operation at the initial time t_0 and the second operation at the time (t_0+2), where the operation O_1 requires 2 time units to finish.

An algorithm will be described by which the states of given unassigned jobs suitable for the given blanks can be determined and, in general, in which it is unnecessary to examine each state of the jobs for each blank.

We recall that if j is a given job, the processing location sequence and processing time sequence are denoted as follows:

$$L_j = (\ell_{j_1}, \ell_{j_2}, \ldots, \ell_{j,nj})$$

$$P_j = (p_{j_1}, p_{j_2}, \ldots, p_{j,nj})$$

We have also introduced the lexicographic order (see Appendix B). By this order, for example, we have 653 > 6432. The labels $L(v_{ij})$ in a state tree can be lexicographically ordered, e.g. 01000 > 00100 ($r=5$). In a state tree, any state (a pendant vertix) corresponds to a label string for which the corresponding vertices constitute the directed path from the root to the pendant vertex. The label strings can also be lexicographically ordered.

The algorithm is described below.

Algorithm 4.4 (Job-Blank Suitability Examining Algorithm)

```
Input    unassigned job set $J$; blank set $B$ of a subschedule $S$; sets
         $R$, $Q$ given by equations (41) and (43), respectively; $B(n,r)$,
           $(n,r) \in R$, $J_b(n,r)$, $\forall\ (n,r) \in R$ and $b \in B(n,r)$; vectors
         $X_b = (x_{b_1}, x_{b_2}, \ldots, x_{b\ell_b})$ and $Y_b = (y_{b_1}, y_{b_2}, \ldots, y_{b\ell_b})$,
         $\forall\ b \in B(n,r)$ and $(n,r) \in R$.

Output   $J_b \subseteq J$, $\forall\ b \in B$ satisfying $\ell_b \geq p$, where $p$ is the minimum
         job length for $J$. For any $j \in J_b$, there exists at least one
         state suitable for $b$; state sets $S_{jb}$ ($j \in J_b$, $b \in B$ and $\ell_b \geq p$)
         for which each state (belonging to job $j$) is suitable for $b$.

Steps    1.  For all $(n,r) \in R$ and $b \in B(n,r)$, $J \leftarrow J_b(n,r)$; $J_b \leftarrow \Phi$;
             repeat 2.
         2.  For all $j \in J$, repeat 2.1 - 2.4;
             2.1  $K_i \leftarrow \Phi$, $i = 1,2,\ldots,r$;
                  $i \leftarrow 1$; $m' \leftarrow 1$
             2.2  $a \leftarrow \sum_{K=1}^{i-1} p_{jK}$;
                  For $m' \leq m \leq r$, if $x_{b,a+m} < \ell_{ji}$ and $y_{b,a+m} > \ell_{ji}$,
                  then $K_i \leftarrow K_i \cup \{a_m\}$, where $a_m$ is a r-bit Boolean
                  vector with a 1 in position $m$ and 0 in other
                  positions. Else $K_i \leftarrow K_i$.
             2.3  If $K_i \neq \Phi$, $i \leftarrow i+1$; else $i \leftarrow n+1$. If $i \leq n$, then,
                  $m' \leftarrow k'$ where $K'$ satisfies $a_{K'} = \min_{a_K \in K_{i-1}} a_K$, then
                  $m' \leftarrow K'$, go back to 2.2.

             2.4  If there exist the label string of length $n$ whose $i$th
                  label is taken from $K_i (i=1,2,\ldots,n)$ and all labels
                  $\ell_1, \ell_2, \ldots \ell_n$ satisfying $\ell_i \geq \ell_{i+1}$ $(i=1,2,\ldots,n-1)$,
                  then $J_b \leftarrow J_b \cup \{j\}$, and let $S_{jb}$ be the set of the
                  label strings. Otherwise, $J_b \leftarrow J_b$ and $S_{jb} = \Phi$.
```

Algorithm 4.4 can determine all states suitable for blank $b \in B$. In fact, given a 2-tuple (n,r) there is a unique state tree $T(n,r)$ corresponding to it. In any tree T, there is one and only one path between any two vertices. For a state tree, each directed path between the root and a pendant vertex has a corresponding label string $\ell_1, \ell_2 \ldots \ell_n$ and ℓ_i satisfying $\ell_i > \ell_{i+1}$ $(i=1,2,\ldots n-1)$. For given $j \in J_b(n,r)$ and $b \in B(n,r)$, a state s of j is suitable for b, if and only if the operation processing locations ℓ_{ji} $(i=1,2,\ldots,n)$ of the job j satisfy inequalities $x_{b,a+m} < \ell_{ji}$ and $y_{b,a+m} > \ell_{ji}$ where $a = \sum_{K=1}^{i-1} p_{jK}$, m being the position of 1 in a_m, a_m being the ith label of the string. Algorithm 4.4 provides a procedure for finding and conserving the directed paths (i.e. the label strings) satisfying the conditions above. In addition, "$b \in B$" has the same meaning as "$b \in B(n,r), \forall (n,r) \in R$", since $B = \bigcup_{(n,r) \in R} B(n,r)$ obviously.

Example. To illustrate Algorithm 4.4, a simple example having one unaassigned job j and one blank b will be considered. Assume that for job j we have

$$n = n_j = 3$$

$$L_j = (7,2,5)$$

$$P_j = (1,1,1)$$

and for blank b,

$$\ell_b = 5, \text{ hence } r = 3$$

$$X_b = (9,6,1,4,6)$$

$$Y_b = (11,8,3,10,14)$$

just like the blank b_3 in Fig. 57. The state tree T(3,3) was shown in Fig. 56.

By Algorithm 4.4, we have the following evolution:

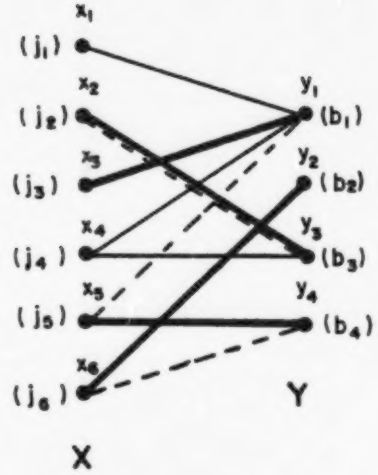
i	a	m	$x_{b,a+m}$	ℓ_{ji}	$y_{b,a+m}$	ℓ_{ji}	k_i
1	0	1	$9 > 7$		$11 > 7$		a_2
		2	$6 < 7$		$8 > 7$		
		3	$1 < 7$		$3 < 7$		
2	1	1	$6 > 2$		$8 > 2$		a_2
		2	$1 < 2$		$3 > 2$		
		3	$4 > 2$		$10 > 2$		
3	2	1	$1 < 5$		$3 < 5$		a_2
		2	$4 < 5$		$10 > 5$		
		3	$6 > 5$		$14 > 5$		

Fig. 58

So, only state $a_2a_2a_2$ is suitable for the blank. Figure 56 shows that it is state $s_7 = (\phi,7,2,5,\phi)$.

For a given unassigned job set J and blank set B, a suitable job set J_b ($\forall$ b ε B) can be determined by Algorithm 4.4 Generally speaking, an unassigned job is suitable for more than one blank, conversely, a blank can be suited to different jobs. A resulting problem is that if there are many different ways to assign the suitable jobs to the blanks, how to find the best one? In other words, how to obtain an assignment such that the number of accepted jobs is the maximum. To answer the question, we introduce the following bipartite graph G(X,Y).

Let X = J and Y = B. There exists an edge xy ($x \varepsilon X$ and $y \varepsilon Y$) in G(X,Y) if and only if x is suitable for y. For example, if $J = \{j_1,j_2,j_3,j_4,j_5,j_6\}$, $B = \{b_1,b_2,b_3,b_4\}$ and $J_{b_1} = \{j_1,j_3,j_4,j_5\}$, $J_{b_2} = \{j_6\}$, $J_{b_3} = \{j_2,j_4\}$, $J_{b_4} = \{j_5,j_6\}$ the resulting bipartite graph G(X,Y) is shown in Fig. 59.



assignment 1
$j_2 \to b_3$
$j_5 \to b_1$
$j_6 \to b_4$
(the dotted line)

assignment 2
$j_2 \to b_3$
$j_3 \to b_1$
$j_5 \to b_4$
$j_6 \to b_2$
(the heavy lines)

Fig. 59

Figure 59 implies that every feasible assignment is a matching of G(X,Y), the desired assignment being the maximum matching. It is a well-known problem to find a maximum matching for a given graph or bipartite graph. The problem can be solved in polynomial time, say, by using the Hopcroft and Karp approach, an $O(n^{5/2})$ algorithm [22], where $n = |X| + |Y|$. It can be found in Appendix B.

If the desired criterion for assigning jobs to blanks is the maximum total job length rather than the maximum number of unassigned jobs, then each edge of G(X,Y) can be assigned a weight, the length of the job associated with the edge. Therefore, a weighted bipartite graph G'(X,Y) is obtained. The bipartite weighted matching problem is also known as the optimal assignment problem and can be solved by the Hungarian method in $O(n^4)$ time ($n = |X| + |Y|$). A version of the method [23], [24] can be found in Appendix B.

4.6 Construction of Exchangeable Job Set Algorithm and Iterative Improvement Procedure

The jobs constituting a T-directed path of the job compatibility digraph D = (V,A) can be done simultaneously. We call the set comprising such jobs a block. Several blocks can constitute a subschedule and the remaining jobs constitute the unassigned job set. Assume that the current subschedule is $S = (B_1, B_2, \ldots B_5)$, that is, S is a sequence of $B_1, B_2, \ldots, B_5$. Let the unassigned job set $J = \{j_1, j_2, \ldots, j_t\}$. A positive number α will be used as a criteria of performance for the blocks. If the ratio between occupied time and idle time in a given block is larger than or equal to α, the block will be regarded as a fair block, else, a poor block.

Suppose that $F = \{B'_1, B'_2, \ldots, B'_{sf}\}$ and $P = \{B''_1, B''_2, \ldots, B''_{sp}\}$ are the sets of fair blocks and poor blocks in a subschedule S, respectively. Obviously, $S = F \cup P$. It is desired to increase the number S_f of fair blocks for a given subschedule S. Let $V(B'') \subset V$ consist of the vertices (jobs) contained by $B'' \varepsilon P$ and $\ell(p)$ be the maximal size of $B'' \varepsilon P$. Exchange $j \varepsilon J$ for j' being in F, and let J_1 be the replacable job set. Then, increasing S_f usually implies that there is a $B'' \varepsilon P$ for which a T-directed path in the subgraph induced by $J_1 \subset V(B'')$ with the length $\ell > \ell(p)$ can be found. It is also possible to find a fair block by increasing occupied time rather than length of T-directed path, or by increasing both of them.

For a given unassigned job $j \varepsilon J$, a collection C(j) will be constructed of which each member may be a single job (exacly, a state of it) in a fair block, it can be substituted by j (exactly, a state of j). If j is a very long job, a member of C(j) may also be several jobs (states) joined together. We call each member of C(j) an exchangeable job (or jobs) of j, and denoted by J(j). The subschedule S is regarded as an array as described in section 4.5, with dimension m x M where m is the number of cranes and M being the makespan of S. It might be assumed that there exists a positive integer $i < M$ such that F, P are the subarrays $\{(S_1,S_2) | 1<S_1<m, 1<S_2<i_0\}$ and $\{(S_1,S_2) | 1<S_1<m, i_0+1<S_2<M\}$ respectively. The following algorithm is used to construct C(j) for $j \varepsilon J$.

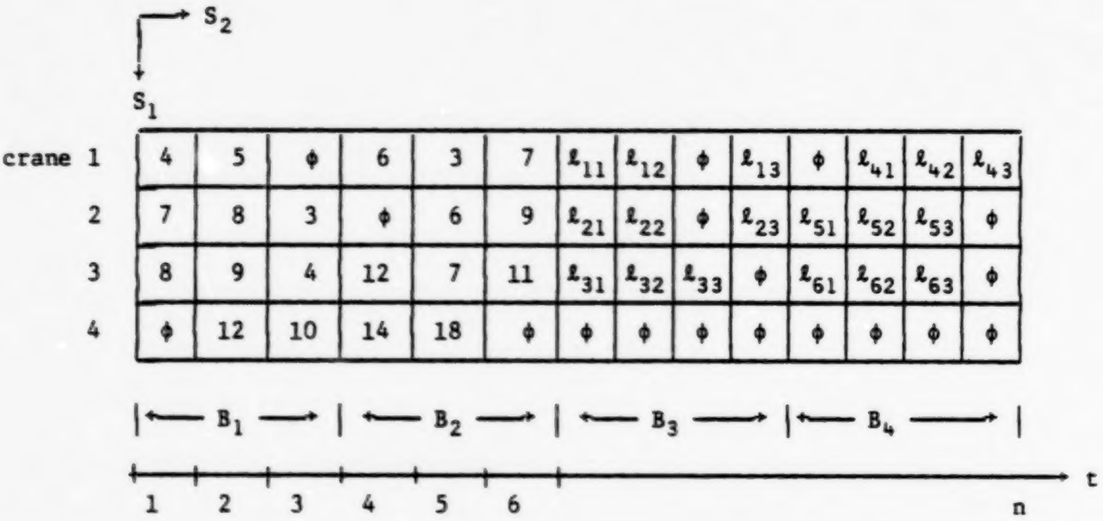
```
Algorithm 4.5  (Construction of Collection C(j) of Exchangeable Job
               Algorithm)

Input     fair block set F, unassigned job set J.

Output    $C(j) = \{J(j) | J(j)$ is the exchangeable job (or jobs) of j,
          $j \varepsilon J_1\}$
          $D(j) = \{b(J(j)) | b(J(j))$ is the cells in S where J(j) is,
          $J(j) \varepsilon C(j)\}$.

Steps     1.  For $j \varepsilon J$, $B(j) \leftarrow \Phi$, $C(j) \leftarrow \Phi$, $D(j) \leftarrow \Phi$, repeat 2;
          2.  For $K_1 = 1,2,\ldots,m$, $K_2 = 1,2,\ldots,i_0-\ell_j$, ($\ell_j$ is the state
              length for j), repeat 2.1 - 2.4;
          2.1 If the sequence of cells $b = ((K_1,K_2),(K_1,K_2+1),\ldots,$
              $(K_1,K_2+\ell_j))$ contains integer of jobs (states)
              $J' = \{j'_1, j'_2, \ldots\}$, then, $B(j) \leftarrow B(j) \cup \{b\}$; else
              $B(j) \leftarrow B(j)$;
          2.2 Constructing sets R', R, Q(n,r), Q according to equations
              (40)-(43) for B(j) generated in 2.1 and $j \varepsilon J$, and
              generating vectors $X_b$, $Y_b$ for $b \varepsilon B(j)$;
          2.3 By algorithm 4.4, $\forall b \varepsilon B = B(j)$ and $j \varepsilon J$, generate the
              following set:
              $J_b = \begin{cases} \{j\} & \text{if j is suitable for b;} \\ \phi & \text{otherwise} \end{cases}$
          2.4 $\forall b \varepsilon B$ and $J_b = \{j\}$, $C(j) \leftarrow C(j) \cup \{J'\}$,
              $D(j) \leftarrow D \cup \{b\}$.
```

Example. We consider the subschedule in Fig. 60 to illustrate the meanings of F, P, B, C, and D. For the subschedule, $\alpha = 1.5$, $i_0 = 6$.



crane														
1	4	5	ϕ	6	3	7	ℓ_{11}	ℓ_{12}	ϕ	ℓ_{13}	ϕ	ℓ_{41}	ℓ_{42}	ℓ_{43}
2	7	8	3	ϕ	6	9	ℓ_{21}	ℓ_{22}	ϕ	ℓ_{23}	ℓ_{51}	ℓ_{52}	ℓ_{53}	ϕ
3	8	9	4	12	7	11	ℓ_{31}	ℓ_{32}	ℓ_{33}	ϕ	ℓ_{61}	ℓ_{62}	ℓ_{63}	ϕ
4	ϕ	12	10	14	18	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ

$J = \{j_1, j_2\}$
$j_1 = (10,5,8)$ $\ell(j_1) = 4$ (the state length for j_1)
$j_2 = (7,5,8)$ $\ell(j_2) = 3$ (the state length for j_2)

The job (4.12) is that filled by the maximum matching algorithm. All the operation processing times are 1 unit.

Fig. 60

For the example, we have:

$$F = \{B_1, B_2\};$$
$$P = \{B_3, B_4\};$$
$$B(j_1) = \{(1,3),(1,4),(1,5),(1,6);\ (2,1),(2,2),(2,3),(2,4);\ (4,2),(4,3),(4,4),(4,5)\}$$
$$B(j_2) = \{(1,1),(1,2),(1,3);\ (1,4),(1,5),(1,6);\ (2,1),(2,2),(2,3);\ (2,4),(2,5),(2,6);\ (4,1),(4,2),(4,3);\ (4,4),(4,5),(4,6)\}$$
$$C(j_1) = \{(6,3,7);\ (12,10),(14,18)\}$$
$$D(j_1) = \{(1,3),(1,4),(1,5),(1,6);\ (4,2),(4,3),(4,4),(4,5)\}$$
$$C(j_2) = \{(6,3,7);\ (6,9)\}$$
$$D(j_2) = \{(1,1),(1,2),(1,3);\ (2,4),(2,5),(2,6)\}$$

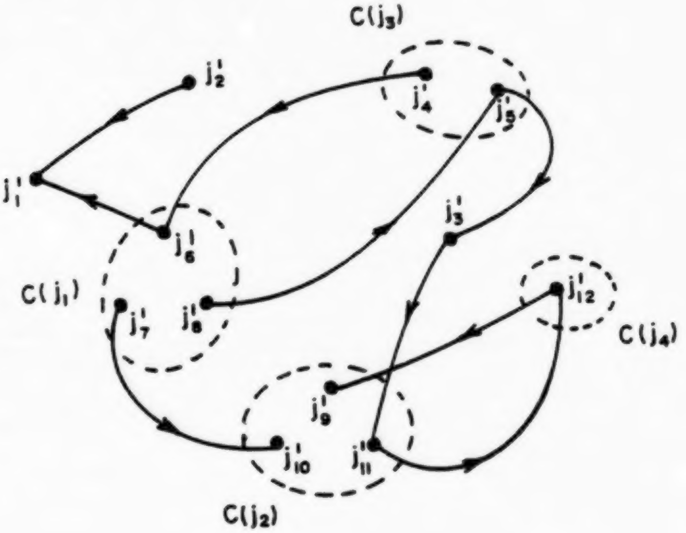
Having generated sets C(j), $j \in J$, in order to construct a fair block consisting of the jobs either belonging to P or being the exchangeable jobs, a directed graph $D' = (V', A')$ must be generated as follows:

1. Specifying the same state length ℓ for all involved jobs;

2. V' consists of the states of 2 types of jobs: (1) job $j \in V'$ belonging to some $B \in P$; (2) there exists $j \in J$ and $J(j)$ such that $j' = J(j) \in C(j)$ where j' may be a combination of more than one job if $J(j)$ is a job group. (Hence, $J(j)$ becomes a job and $C(j)$ becomes a set). For example, for B_3 of the example above, D' has 6 vertices $(\ell_{11},\ell_{12},\ell_{13})$, $(\ell_{21},\ell_{22},\ell_{23})$, $(\ell_{31},\ell_{32},\ell_{33})$ and (6,3,7), (12,10,14,18), (6,9).

3. If states S_1, S_2 belong to the jobs j'_1, j'_2 respectively, $j'_1 \in C(j_1)$, $j'_2 \in C(j_2)$, $j_1 \neq j_2$ and S_1, S_2 have the relation $S_1 \underset{s}{\rightarrow} S_2$, then $(S_1,S_2) \in A'$.

Such a directed graph is shown in Fig. 61. For evidence, the job occurring in more than one C(j) is regarded as a different job (e.g. (6,3,7) is such a job in the example above).



j'_1, j'_2, j'_3 $\in$ some $B \in P$,
j'_4 — j'_{12} $\in F$,
$J = \{j_1,j_2,j_3,j_4\}$

Fig. 61

In general, any realistic digraph $D' = (V',A')$ is very complex. For simplicity, for D' in Fig. 61, assume that one job has only one state. Note that, for some jobs (states), relation $\underset{s}{\rightarrow}$ is probably transitive, but this case has not been considered in Fig. 61.

D' has a property similar to that of state compatibility digraph D, that is, the vertex set can be divided into subsets in each of which any two vertices are not adjacent. However, the subsets C(j)'s of V' only satisfy condition (34) in section 4.2. Condition (33) is not met because the same j' can occur in different C(j)'s. The fact will cause some trouble, namely, there may be two kinds of directed path (not necessarily T-directed), as shown in Fig. 62.

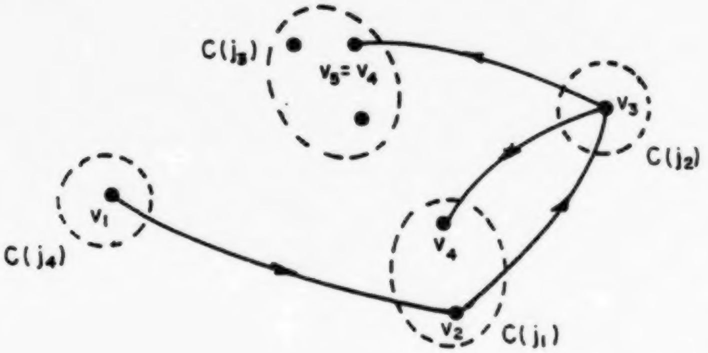


Fig. 62

There are two directed paths in the graph: $p_1 = v_1v_2v_3v_4$ and $p_2 = v_1v_2v_3v_5$ both of which consist of the same jobs (states), say j'_1, j'_2, j'_3, j'_4. p_1 is unacceptable, because it is not allowed that both job j'_2 and job j'_4 be replaced by the same job j_1. p_2 is acceptable, because they are replaced by different jobs j_1, j_3. Then, for the paths consisting of the same vertices how to identify the acceptable path? If general directed paths are considered, it is indeed a problem (it can be solved by extending N-matrix to N_K-matrix, K is the degree of the matrix). However, to find a T-directed path, there is no problem. Actually, it is impossible that p_1 is a T-directed path, because both (v_2,v_4) and (v_4,v_2) do not belong to A'. Therefore, we can claim that if the same job belonging to different C(j)'s is regarded as different vertices, then, Algorithm 4.1 is available for finding T-directed paths, of digraph D', as long as A(D'), $B^-(D')$ are defined according to Definition 4.10.

Example. Consider D' in Fig. 63.

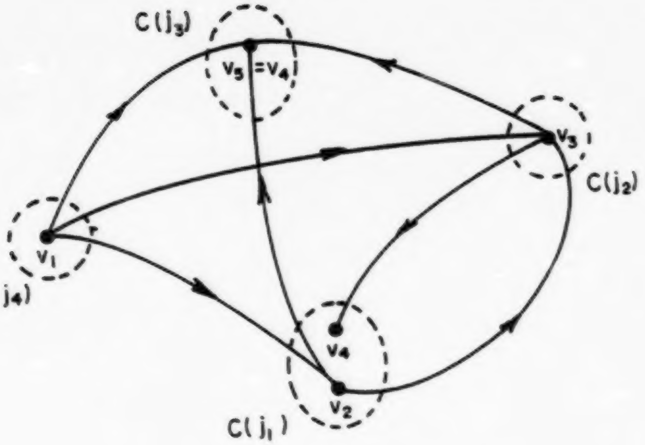

C(j3)
v5 = v4
v3
C(j2)
v1
C(j4)
v4
v2
C(j1)

Fig. 63

We have:

$$A(D') = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & - & \{1\},2 & \{1\},3 & - & \{1\},5 \\ 2 & - & - & \{2\},3 & - & \{2\},5 \\ 3 & - & - & - & \{3\},4 & \{3\},5 \\ 4 & - & - & - & - & - \\ 5 & - & - & - & - & - \end{array}$$

$$B(D') = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & - & \{1\},2 & \{1,2\},3 & - & \{1,2,3\},5 \\ 2 & - & - & \{1,2\},3 & - & \{1,2,3\},5 \\ 3 & - & - & - & \{3\},4 & \{1,2,3\},5 \\ 4 & - & - & - & - & - \\ 5 & - & - & - & - & - \end{array}$$

$A^{(2)}(D') = A(D')$.

$$A^{(3)}(D') = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & - & - & \{1,2\},3 & - & \{1,2\},5 \\ 2 & - & - & - & - & \{1,3\},5 \\ 3 & - & - & - & - & - \\ 4 & - & - & - & - & - \\ 5 & - & - & - & - & - \end{array} \qquad (44)$$

$$A^{(4)}(D') = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left(\begin{array}{ccccc} - & - & - & - & \{1,2,3\},5 \\ - & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \end{array} \right) \end{array} \qquad (44)$$

$$A^{(5)}(D') = 0.$$

It is clear that there is a unique T-directed 4-path $v_1 v_2 v_3 v_5$ in D'.

Now we set up the iterative improvement algorithm as follows. Let p be a path of D', the vertex set of p is denoted by V_p. The following auxiliary algorithms are needed.

Algorithm 4.6.1

```
1.  $J_1 \leftarrow V(B) \cup C$;
2.  Construct $D' = (V', A')$, where $V' = J_1$;
3.  Find the T-directed m-paths of D', where m is the number of
    cranes. If there is no m-path, the longest path will be
    found. Let $\ell$ be the path length ($\ell < m$).
```

Algorithm 4.6.2

```
1.  $I \leftarrow I - B$, $I_1 \leftarrow I_1 - B$;
2.  $\forall$ $C(j) \in V_p \cap C$, exchange j for C(j);
3.  Produce the new schedule S, the position of j in S is given
    by set D(j), in Algorithm 4.5;
4.  $J_1 \leftarrow \{j \mid j \bar{\in} V_p \cap (B \cup J_1)\}$;
5.  For $j \in J_1 \cap B$, find C(j), D(j) by Algorithm 4.5;
    $C \leftarrow \{C(j), j \in J_1\}$, $D \leftarrow \{D(j), j \in J_1\}$.
```

Algorithm 4.6 (Iterative Improvement Algorithm)

Input fair block set F, poor block set p, maximum size $\ell(p)$ of the blocks in P, $C = \{C(j) \mid j \in J\}$ and $D = \{D(j) \mid j \in J\}$ where J is the unassigned job set (C, D are produced by Algorithm 4.5).

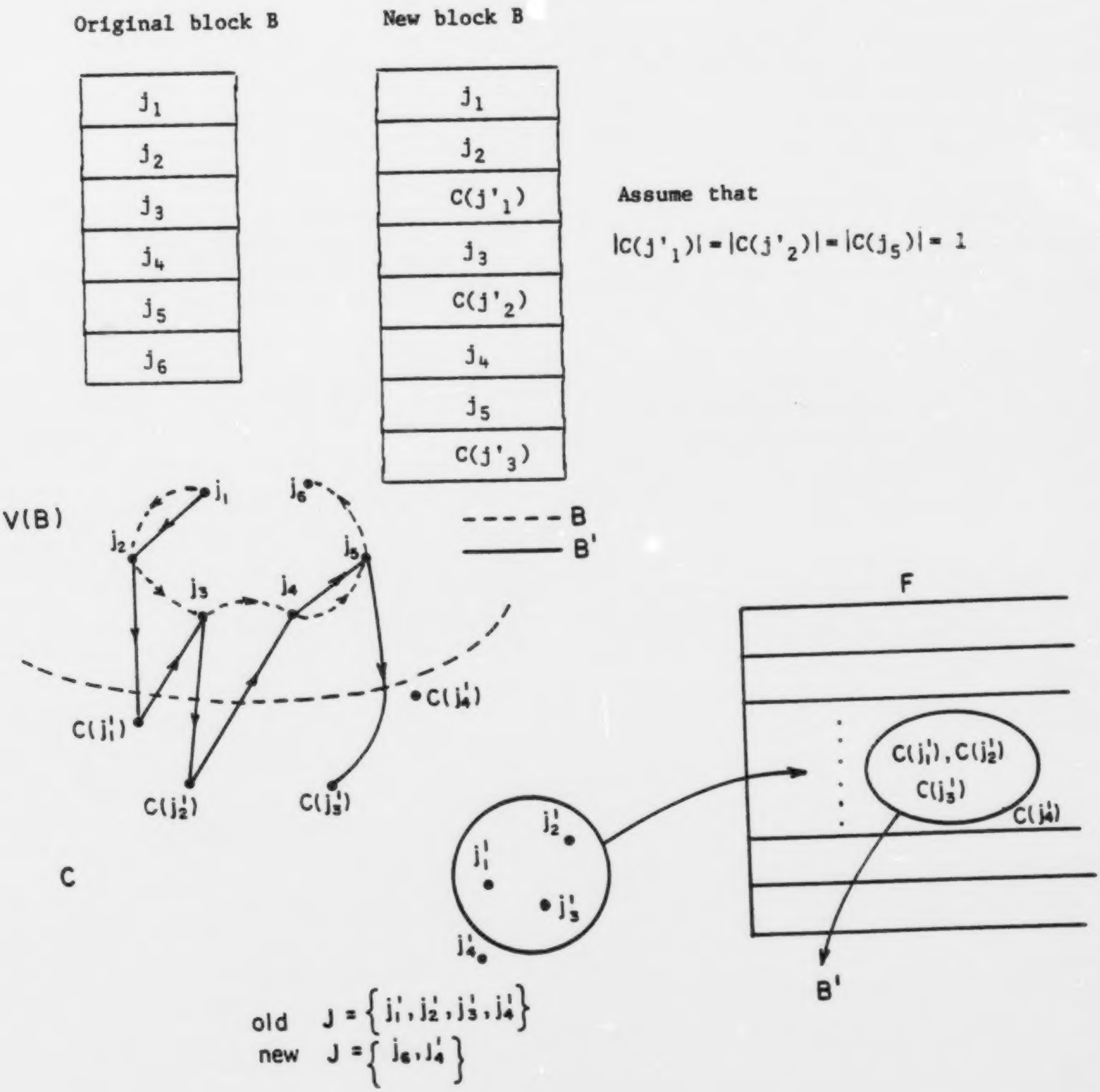
Output improved subschedule S, its makespan M, unassigned job set J.

Steps

```
1.  $I \leftarrow P$;
2.  If $I \neq \phi$, $I_1 \leftarrow I$, pick $B \in I$ arbitrarily and do
    Algorithm 4.6.1; If $\ell > \ell(P)$, do Algorithm 4.6.2; go to
    3; Otherwise, $I \leftarrow I - B$, go back to 2; If $I = \Phi$, go to 4;
3.  If $I_1 \neq \phi$, pick $B \in I_1$ arbitrarily and do Algorithm 4.6.1;
    If $\ell > \ell(P)$, do Algorithm 4.6.2, go back to 4.6.3;
    Otherwise, $I_1 \leftarrow I_1 - B$, go to 3; If $I_1 = \Phi$, go back to 2;
```

4. Calculate the makespan M of S, $J \leftarrow J_1$;
5. Repeat 1-4 until no improvement can be done.

It is clear that the subschedule $S = F \cup P$ gains an improvement in total idle time and a reduction in number of unassigned jobs, as long as Algorithm 4.6.2 is performed. The fact can be shown in Fig. 64.


Original block B
New block B
j_1
j_2
j_3
j_4
j_5
j_6
j_1
j_2
$C(j'_1)$
j_3
$C(j'_2)$
j_4
j_5
$C(j'_3)$
V(B)
B
B'
C
$C(j'_1)$
$C(j'_2)$
$C(j'_3)$
$C(j'_4)$
j'_1
j'_2
j'_3
j'_4
F
$C(j'_1), C(j'_2)$
$C(j'_3)$
$C(j'_4)$
B'

Assume that

$|C(j'_1)| = |C(j'_2)| = |C(j_5)| = 1$

old $J = \{j'_1, j'_2, j'_3, j'_4\}$

new $J = \{j_6, j'_4\}$

Fig. 64

4.7 The PCMI Method

The existing method to solve crane scheduling problems, including the CCH method described in the previous section, are designed for special cases of the problems, and the resulting schedules have special structuring. The constraints associated with these special problems and required to apply the methods, are very strong. Even so, the problems are still NP-complete. Therefore, one cannot expect to find an optimal solution in polynomial time. In other words, any algorithm can only construct a feasible solution in polynomial time but will not guarantee an optimal solution. In general, the more the structuring of resulting schedules are restricted, the smaller are the subspaces of the searched feasible solutions, the further the solutions are from the optimal solutions.

The special constraints imposed on crane scheduling problems by the previous methods are relative to three parameters: m (no. of cranes), n_j (no. of operations of job j) and p_{ji} (operation processing time) or p_j (job processing time). The structuring of schedules (subschedules) can be divided into two cases:

1. No idleness between operations. There are three forms for the case, batching, meshing and others (see Fig. 65).
2. Idleness between operations is allowed, namely, the general structuring form (see Fig. 65).

The permissible forms of idleness can be considered as special structuring forms.

Our present purpose is to develop a procedure for constructing a schedule with general structuring forms for the crane scheduling problems without the constraints on m, n_j and p_{ji}, p_j, using the algorithms described previously. The procedure is called the PCMI method.

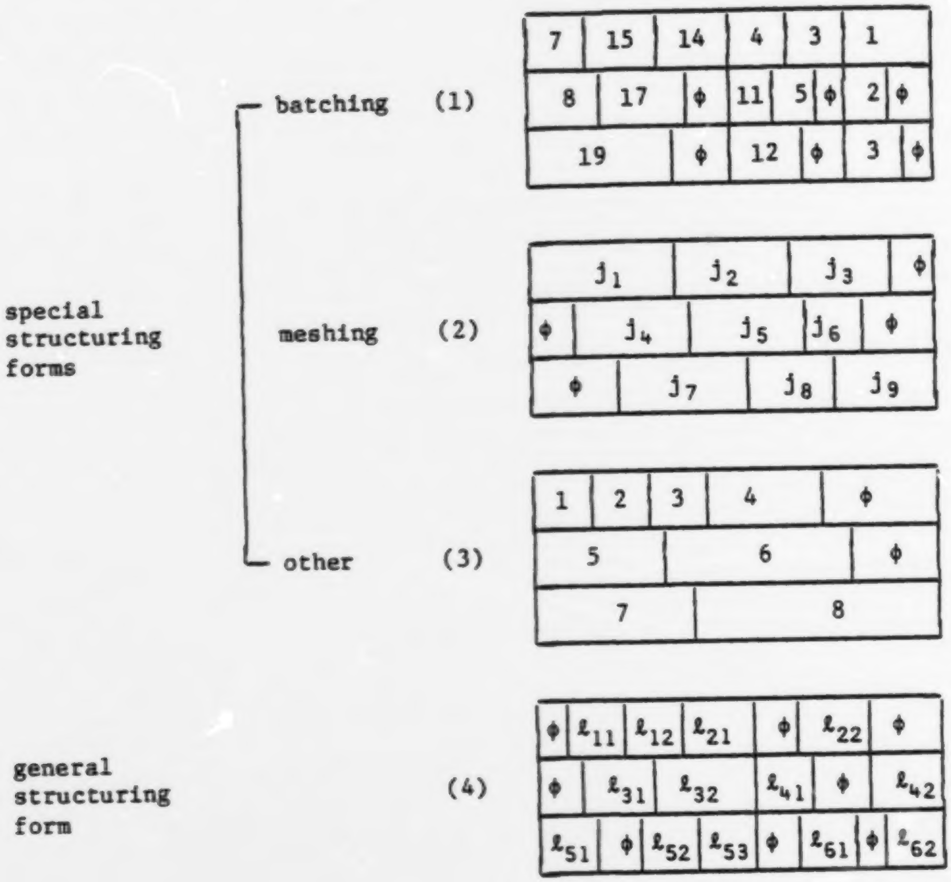

special structuring forms
batching (1)
7 15 14 4 3 1
8 17 ϕ 11 5 ϕ 2 ϕ
19 ϕ 12 ϕ 3 ϕ
meshing (2)
j_1 j_2 j_3 ϕ
ϕ j_4 j_5 j_6 ϕ
ϕ j_7 j_8 j_9
other (3)
1 2 3 4 ϕ
5 6 ϕ
7 8
general structuring form
(4)
ϕ ℓ_{11} ℓ_{12} ℓ_{21} ϕ ℓ_{22} ϕ
ϕ ℓ_{31} ℓ_{32} ℓ_{41} ϕ ℓ_{42}
ℓ_{51} ϕ ℓ_{52} ℓ_{53} ϕ ℓ_{61} ϕ ℓ_{62}

Fig. 65

Some algorithms for solving crane scheduling problems have been presented in [3,4]. The constraints and structuring forms of the schedules for the algorithms in [3,4], for the CCH method and the PCMI method are listed in Fig. 66.

We give an outline of PCMI method before the entire procedure is described. In the PCMI method, states will be considered instead of jobs. The idleness between operations is allowed in a state, therefore, it is possible to construct a schedule with a general structuring form. Assume that the state compatibility digraph D = (V,A) has been established. The first stage of the PCMI method is to find the T-directed paths of D with given length K_1. Each of the paths determines K_1 states (any two not belonging to the same job, hence, K_1 jobs) which can be done simultaneously.

Method (or algorithm)	Constraints				Structuring form of schedule
	m	n_j	p_{ji}	p_j	
Algorithms Z_3 [3]	arbitrary	$n_j = 1$ $\forall j \in J$	arbitrary	arbitrary	idleness [(3),Fig.65]
Batching Procedure [4]	arbitrary	$n_j = 2$ $\forall j \in J$	$p_{ji} = \tau$ $\forall j \in J, i=1,2$	$p_j = 2\tau$ $\forall j \in J$	batching [(1),Fig.65]
Mixed Procedure [4]	$m = 2$	$n_j = 2$ $\forall j \in J$	$p_{ji} = \tau$ $\forall j \in J, i=1,2$	$p_j = 2\tau$ $\forall j \in J$	batching & meshing [(1)(2),Fig.65]
CCH Method	arbitrary	arbitrary	arbitrary	$p_j = p$ $\forall j \in J$	batching [(1),Fig.65]
PCMI Method	arbitrary	arbitrary	arbitrary	arbitrary	idleness [(4),Fig.65]

(τ is the time unit)

Fig. 66

There may be many different T-directed K_1-paths in D. Some of them are vertex-joint paths, and some are vertex-disjoint. Since every vertex is a state, the vertex-disjoint paths are also state-disjoint paths. The state-disjoint paths are not necessarily job-disjoint paths, since some states in different paths may belong to the same job. It is easy to obtain the corresponding job paths from state paths by means of a converting list between the numbers of states and numbers of jobs. For example, if there are 4 jobs j_1, j_2, j_3, j_4 and they have 3, 6, 3, 10 states respectively, the list is as follows:

1,1,1,2,2,2,2,2,2,3,3,3,4,4,4,4,4,4,4,4,4,4

It shows that, say, the 5th state s_5 belongs to j_2, s_{11} belongs to j_3, and so on. Using the list, each state-path can be transformed into a job path. Thus, we construct a graph $G' = (V',E')$ to express the job-disjoint relation of the paths. G' is such a graph in which every vertex is a job path, and $(v_i,v_j) \in E'$ iff P_i, P_j are job-disjoint paths.

If there is a group of paths of which any two are job-disjoint, then, these job-disjoint paths (they are all T-transitive) can be used to construct a subschedule, and each path gives a group of jobs (namely, a block) which can be done simultaneously. Hence, it is expected to find the paths which are as job-disjoint as possible. This leads us to finding the maximum cliques of undirected graph G'.

Algorithm 4.2 provides a procedure for finding a clique with size K_2. By invoking the algorithm until $A^{(i)}$ becomes zero-matrix, the maximum cliques will be obtained.

Assume that there are 5 job-disjoint paths, each of which is called a block as in subsection 4.6. Having set the value of α, the blocks can be divided into fair blocks F and poor blocks P. Generally speaking, the idle time interval is not too long in a fair block. However, if the number of fair blocks is rather large, long idle intervals can occur for two reasons. First, combination of fair blocks can result in producing a long idle interval . The example shown in Fig. 67 illustrates the case.

B_1							B_2								B_3					
			ϕ			ϕ			ϕ			ϕ				ϕ				
					ϕ	ϕ	ϕ	ϕ	ϕ								ϕ			
		ϕ			ϕ							ϕ	ϕ	ϕ	ϕ	ϕ				

(other regions are busy times)

Fig. 67

Second, long idle time intervals can occur by transforming a subschedule into an active one. We know that a schedule is optimal only if it is active; and an active schedule is such that no operation can be started earlier without either delaying some other operation or violating the constraints of the crane scheduling problem [25]. Figure 68 is an illustration of this case.

B_1							B_2						
ℓ_{11}	ℓ_{12}	ℓ_{13}	ϕ	ℓ_{14}	ϕ	ℓ_{15}	ℓ_{21}	ℓ_{22}	ϕ	ℓ_{23}	ℓ_{24}	ϕ	ℓ_{25}
ϕ	ϕ	ℓ_{31}	ℓ_{32}	ℓ_{33}	ℓ_{34}	ϕ	ϕ	ϕ	ℓ_{41}	ℓ_{42}	ℓ_{43}	ℓ_{44}	ℓ_{45}
ℓ_{51}	ℓ_{52}	ϕ	ϕ	ℓ_{53}	ℓ_{54}	ℓ_{55}	ℓ_{61}	ℓ_{62}	ℓ_{63}	ℓ_{64}	ϕ	ϕ	ϕ

unactive schedule

ℓ_{11}	ℓ_{12}	ℓ_{13}	ϕ	ℓ_{14}	ϕ	ℓ_{15}	ℓ_{21}	ℓ_{22}	ϕ	ℓ_{23}	ℓ_{24}	ϕ	ℓ_{25}
ℓ_{31}	ℓ_{32}	ℓ_{33}	ℓ_{34}	ϕ	ϕ	ϕ	ϕ	ϕ	ℓ_{41}	ℓ_{42}	ℓ_{43}	ℓ_{44}	ℓ_{45}
ℓ_{51}	ℓ_{52}	ℓ_{53}	ℓ_{54}	ℓ_{55}	ℓ_{61}	ℓ_{62}	ℓ_{63}	ℓ_{64}	ϕ	ϕ	ϕ	ϕ	ϕ

active schedule

Fig. 68

At the same time, there are still some unassigned jobs. In general, for each poor block (or some fair blocks with long idle time intervals), the number of jobs is less than m, m is the number of cranes. However, each unassigned job cannot be inserted into any one of the blocks, otherwise the job would have been included in the block. However, it is possible to insert an unassigned job into a blank (idle time interval). Assume B is the blank set and J is unassigned job set, by Algorithm 4.4, $J_b \subseteq J$, $\forall$ $b \in B$ can be obtained. Then, a bipartite graph G(X,Y) can be constructed and an optimal assignment between unassigned jobs and blanks can be determined by the Hoperoft and Karp approach.

A job not previously assigned to a fair block F can enter F through exchanging. Job exchanging results from reconstruction of poor blocks and may reduce makespan. The process can be repeated time and time again using Algorithm 4.6. In other words, it is possible that a schedule can be improved iteratively. In Algorithm 4.6, an actual exchange is made if and only if the makespan can be reduced. However, it does not mean that a schedule with smaller makespan can always be found if it is not optimal. That is, it is not assured that Algorithm 4.6 converges to an optimal solution.

Next, we will sum up the discussion above which comprises the PCMI Method.

Preliminaries

1. Set the value of state length ℓ. Given a job j and state length ℓ, the number n_s of states belonging to j depends on n_j and $r = \ell - p_j + 1$, and increases exponentially with n_j or r. ℓ should not be too large, otherwise, the total number of states will become enormous.

2. Construct state set S(j) for each job j. S(j) is the set of pendant vertices of the state tree $T(n_j, r)$, with every pendant vertex corresponding to a label string. By Algorithm 4.3, all label strings, hence all states can be produced.

3. Construct state compatibility digraph D = (V,A). D is stored as an adjacency matrix. The vertices are numbered according to positive integers $1, 2, \ldots, |V| = N_s$, where N_s is the total number of states: $N_s = \sum_{j \in J} |S(j)|$.

PCMI Method The T-Directed Path - Clique - Maximum Matching - Iterative Improvement Method

Phase P Producing T-Directed Paths of Length K1 in Digraph D

Input The adjacency matrix of the state compatibility digraph D = (V,A), path length K1, state-job converting list L: no. of state → no. of job.

Output T-directed state K1-paths, if there is no path of length K1, the longest paths are produced and their length assigned to K1; the job-disjoint relation graph G.

Steps
1. $K \leftarrow 2$;
2. Calculate N-matrix $A^{(K+1)}(D)$ by Algorithm 4.1 (the step 3 of Algorithm 4.1);
3. If $A^{(K+1)}(D) = 0$, then $K1 \leftarrow K$; produce matrix A by replacing each 2-tuple $(\{i, i_2, \ldots, i_{K-1}\}, i_K)$ of $A^{(K)}(D)$ by 2-tuple $(\{L(i_1), L(i_2), \ldots, L(i_{K-1})\}, L(i_K))$, where $L(i)$ is ith entry of L;
4. In the case of $A^{(K+1)}(D) \neq 0$, if $K < K1-1$, $K \leftarrow K+1$ go back to 2; otherwise, produce matrix A by replacing each 2-tuple $(\{i_1, i_2, \ldots, i_K\}, i_{K+1})$ of $A^{(K+1)}(D)$ by 2-tuple $(\{L(i_1), L(i_2), \ldots, L(i_K)\}, L(i_{K+1}))$;
5. Construct job-disjoint relation graph G taking the state-disjoint paths as vertices, and uv being an edge if and only if p_1, p_2 are job-disjoint paths, where p_1, p_2 are the 2-tuples of A and corresponding to vertices u,v. Take positive integers $1, 2, \ldots, N_p$ as the labels of the vertices of G N_p is the total number of state-disjoint paths.

Phase C Producing Cliques of Size K2 in Graph G

Input The job-disjoint relation graph G, clique size K2, positive number α.

Output Subschedule $S = (B_1, B_2, \ldots, B_s)$, fair block set $F = \{B'_1, B'_2, \ldots, B'_{sf}\}$, poor block set $P = \{B''_1, B''_2, \ldots, B''_{sp}\}$, unassigned job set J, blank set B.

Steps
1. $K \leftarrow 2$;
2. Calculate $A^{(K)}(G)$ by Algorithm 4.2 (or step 3 of the algorithm);
3. If $A^{(K)}(G) = 0$, then $K2 \leftarrow K-1$; $K \leftarrow K-1$;
   If $A^{(K)}(G) \neq 0$ and $K < K2$, go back to 2;
4. If $A^{(K)}(G) \neq 0$ and $K = K2$, then each 2-tuple $(\{i_1, i_2, \ldots, i_{K-1}\}, i_K)$ of $A^{(K)}(G)$ is a K2-clique of G, and each i, $i = i_1, i_2, \ldots, i_K$ is a T-directed state path, i.e., a block. Calculate the total idle time of each clique and the clique with the minimal total idle time is selected;
5. Calculate the ratio between occupied time and idle time for each block and divide all blocks contained by the selected clique into fair blocks set F and poor blocks set P; Construct subschedule S in which fair blocks precede poor blocks; the remaining jobs constitute the unassigned job set J;
6. Generate the active subschedule from S(denoted by the same symbol S); produce blank set B.

Phase M Filling Unassigned Jobs in Blanks by Finding Maximum Matching

Input: Subschedule S, unassigned job set J, blank set B.

Output: Resulting subschedule S from filling unassigned jobs in blanks, maximum matching $\{(j,b) \mid \text{job } j \text{ is assigned to blank } b\}$; unassigned job set $J_1 \subseteq J$.

Steps

1. Generate sets R, Q according to equations (41), (42); Generate sets $B(n,r)$, $J_b(n,r)$, vectors X_b and Y_b (see section 4.5);
2. Produce $J_b \subseteq J$, $\forall\ b \in B$. For any $j \in J_b$, there is one state at least to be suitable for b; produce S_{jb} ($j \in J_b$, $b \in B$) of which each state (belonging to j) is suitable for b; Both J_b and S_{jb} are produced by Algorithm 4.4;
3. Construct bipartite graph $G(X,Y)$, $X = J$, $Y = B$, xy is an edge if and only if x is suitable for y;
4. By Hoperoft and Karp method, find the maximum matching $M = \{(j,b) \mid j \text{ is assigned to } b\}$ of $G(X,Y)$;
5. For each $(j,b) \in M$, fill job j in blank b, and produce a new subschedule S. The remaining jobs in J constitute the new unassigned job set, denoted by J_1.

Phase I Iterative Improvement Procedure

Input: Fair blocks F and poor blocks P obtained in Phase C, unassigned job set J_1 obtained in Phase M, state lengths $\ell(j)$, $j \in J_1$.

Output: Schedule S for given crane scheduling problem, makespan M.

Steps

1. By Algorithm 4.5, produce the exchangeable job set $C(j)$ for each $j \in J_1$ and the set of the locations of these exchangeable jobs;
2. Produce a new subschedule S and new unassigned job set J by Algorithm 4.6;
3. If $J \neq \Phi$, find a longest path $p = j_1, j_2, \ldots j_K$ of subgraph D_J of the state compatibility digraph D, induced by J (the length K of the path is less than m, the number of cranes, obviously);
4. $D_J \leftarrow D_J - \{j_1, j_2, \ldots, j_K\}$; $J \leftarrow J - \{j_1, j_2, \ldots, j_K\}$;
5. $S \leftarrow S \cup B_p$, where B_p is the block produced by p; go back to 3; if $J = \Phi$, calculate the makespan M of S.

Phases P, C, M and I are the main body of the PCMI method. To apply the PCMI method to an actual problem, the method must be combined with computational considerations which will be discussed in the next subsection.

The algorithms comprising the PCMI method are all polynomial. For given constants K1 and K2, Algorithms 4.1 and 4.2 are polynomial because calculating of matrices $A^{(K1)}(D)$ and $A^{(K)}(G)$ are only required. Algorithm 4.6 (iterative improvement procedure) is polynomial for an iterative loop since Algorithm 4.1 is polynomial. The Hoperoft and Karp method is known to be effective and it is clear that Algorithms 4.3, 4.4, 4.5 only require polynomial computing time.

4.8 Computational Considerations and Data Structures

1. Grouping of Jobs

For given jobs, some of them may be long and others may be short. If long jobs and short jobs are handled simultaneously, it will be difficult to select a reasonable state length ℓ. First, it is necessary that $\ell > \max p_j$. If $\ell = \max p_j$, there may be many states belonging to a short job but, on the other hand, for the long jobs, there may be only one or a few states. The computing time required for handling the job increases with the number of states. If the number of states is smaller, the feasible solution space searched by the method is smaller and the resulting schedule is further from the optimal schedule. Therefore, a job set including both long jobs and short jobs is undesirable. In this case, a practical course of action is to divide the job set into subsets so that for each subset the job lengths and numbers of operations are similar.

2. Selection of State Length

The jobs of the same group have the same state length ℓ. The number n_s of states belonging to job j depends on n_j and $r = \ell - p_j + 1$. When r is fixed, $n_j = 1,2,3$ or n_j is fixed, $r = 1,2,3$, or $n_j = 1,2,3$ and $r = 1,2,3$, the values of n_s are shown in Fig. 69.

n_j	r	n_s
1	r	r
2	r	$\frac{r(r+1)}{2}$
3	r	$\frac{r(r+1)(r+2)}{6}$

r	n_j	n_s
1	n_j	1
2	n_j	$n_j + 1$
3	n_j	$\frac{(n_j-2)(n_j+5)}{2} + 6$

n_j \ r	1	2	3
1	1	2	3
2	1	3	6
3	1	4	10

Fig. 69

It appears that for fixed r, n_s is $O(r^{n_j})$ and for fixed n_j, n_s is $O(n_j^{r-1})$. For actual problems we have met, $n_j < 3$, and it is appropriate to make $n_s < 10$.

3. Selection of K1, K2

K1 is the length of T-directed path. Commonly, K1 is set to m (the number of cranes). If there does not exist an m-path, K1 can be reduced automatically and a longest path will be found. K2 is the clique size, it can increase gradually from 2. In applying the PCMI method to an actual problem, it may be good to test several combinations of K1 and K2 and pick the best values.

4. Repetition of Phase P and Phase C

It is not necessary that Phase C be immediately followed by Phase M. If there are many jobs, Phase P and Phase C can be applied once again after Phase C. Phase M is better started when the number of unassigned jobs is not large. Most jobs should be assigned through Phase P and Phase C. In reality, if only a few jobs are assigned, there are insufficient blanks in the fair blocks.

5. Applying Phase M and Phase I Alternately

In some cases, one cannot fill blanks with any unassigned jobs, not for lack of blanks but because there are no suitable unassigned jobs. After Phase I, a new unassigned job set is created. At this time, it may be possible to fill the blanks with some of these unassigned jobs. Therefore, if there are blanks, it may be good to perform Phase M after each iteration.

Now let us turn to the data storage and data transferring problems relative to the PCMI method. States are the input and a schedule is the output. From input to output, the transferring of data is expressed by the following chain:

$$\text{state} \rightarrow \text{graph} \rightarrow \text{matrix} \rightarrow \text{path} \rightarrow \text{schedule}$$

They can be discussed respectively as follows:

1. State

A state can be expressed in two ways:

Expression of l vector The geometrical form of this expression is that which we have been using in this report. The digital form is very simple. For example, assume that for job j: $n_j = 3$, $P_j = (2,1,3)$, $L_j = (3,8,5)$, and the state length $\ell = 8$. If the processing location for the virtual operation is denoted by a large number, say 99, then, we have: $s_1 = 9999030308050505$, $s_2 = 0303999908050505$, and so on.

Expression of 2 vectors In a state tree, a state can be expressed by a label string. Each label a_j is an r-bit Boolean vector, only the jth component being 1. Thus, a_j is uniquely determined by j, a label string being a vector. The vector combined with the processing time vector (sequence) P_j can form a state. For the example above, we have $r = 8-6+1 = 3$. The label strings of states s_1, s_2 are (3,3,3) and (1,3,3), respectively. The P_j is the same for all states belonging to the same job, and here $P_j = (2,1,3)$.

2. Graph

There are four graphs (or digraphs) involved in the PCMI method:

$D = (V_1,A)$ the state compatibility digraph;
$G = (V_2,E)$ the job-disjoint relation graph;
$G = (X,Y)$ the job-blank suitability bipartite graph;
$D' = (V',A')$ the replacable state compatibility digraph

A graph (or digraph) can be represented by either a matrix (adjacency matrix or incidence matrix) or the lists (adjacency lists or incidence lists). From storage considerations, it is usually more advantageous to use adjacency lists than the adjacency matrix to store a sparse graph. However, from operations considerations, we would rather use matrices than lists. Moreover, we can directly generate N-matrices A(D) (or A(G), A(D')) rather than the adjacency matrix, because of the fact that for a given adjacency matrix of D, N-matrix A(D) can be uniquely determined, and vice versa.

Graph operations are often encountered, for example removing some vertices and the edges incident to them. To do this, elimination of the rows and columns corresponding to the vertices is needed.

3. Matrix

In PCMI method, the matrices including the immediate results are all N-matrices, except the matrix representing the bipartite graph G(X,Y). All the N-matrices are stored as sparse matrices, that is, only the non-zero elements are stored. In fact, it is impossible to store the entire N-matrix $A^{(K)}(D)$ (or $A^{(K)}(G)$), because there may be n^K 2-tuple in an element for the worst case, where $n = |V_1|$. As a result, matrix operations have to be done in sparse matrix form.

4. Path

In the method, a K-path is a 2-tuple of N-matrix $A^{(K+1)}(D)$, and a clique is a group of paths. Each path consists of states. If there is need to produce the block corresponding to the path, the states will be found and used to form a 2-dimensional array.

5. Schedule and Subschedule

A schedule (subschedule) is a 2-dimensional array. The initial subschedule comprises blocks. After filling and iteration, the resulting subschedule can no longer be partitioned into blocks. In general, a schedule (a solution of the crane scheduling problem) possesses the general structuring form shown in (4) of Fig. 65.

Since the selection of a particular data structure can noticeably affect the speed and efficiency of an algorithm, decisions about the representation must incorporate a knowledge of the algorithms to be applied. Conversely, the choice of an algorithm may depend on how the data is initially given. There are 7 algorithms to be applied in PCMI method. We are not sure the data structures discussed above are the most advantageous forms for each algorithm.

5.0 REFERENCES

1. Graefe, P.W.U., Nenonen, L.K., Strobele, K. "Simulation of Combined Discrete and Continuous Systems on a Hybrid Computer". Simulation, pp. 129-137, (May 1974).

2. Lieberman, R.W. "Scheduling Under Interference Constraints". Ph.D. Thesis, University of Toronto (October 1979).

3. Lieberman, R.W., Turksen, I.B. "Crane Scheduling Problems". AIIE Transactions, 13, 304-311 (December 1981).

4. "Two-operation Crane Scheduling Problems". IIE Transactions, 14, 147-155 (September 1982).

5. Mellor, P. "A Review of Job-Shop Scheduling Problem". Ops. Res. Q., 17, 161-171 (1966).

6. Hu, T.C. "Parallel Scheduling and Assembly Line Problems". Operations Res. 9, 841-848 (1961).

7. Ullman, J.D. "NP-Complete Scheduling Problems". J. Comput. System Sci. 10, 384-393 (1975).

8. "Complexity of Scheduling Problems". In E.G. Coffman (ed.) Computer and Job-Shop Scheduling Theory. John Wiley and Sons, New York, Chapter 4.

9. Lawler, E.L. "Combinatorial Optimization: Networks and Matroids". Holt, Rinehart and Winston, New York.

10. Berge, C. "Graph and Hypergraph". Chapter 16. Amsterdam, North-Holland, 1973.

11. Karzanov, A.V. "Determining the Maximum Flow in a Network by the Method of Preflows". Soviet Math. Dokl., 15, 434-437 (1974).

12. Golumbic, M.C. "The Complexity of Comparability Graph Recognition and Coloring". Computing 18, 199-208 (1977).

13. Fulkerson, D.R.
Gross, O.A. "Incidence Matrices and Interval Graphs". Pacific J. Math. 15, 835-855 (1965).

14. Dirac, G.A. "On Rigid Circuit Graphs". Abh. Math. Sem. Uni. Hamburg, 25, 71-76 (1961).

15. Gilmore, P.C.
Hoffman, A.J. "A Characterization of Comparability Graph and of Interval Graphs". Can. J. Math. 16, 539-548 (1964).

16. Golumbic, M.C. "Comparability Graphs and a New Matroid". J. Comb. Th. Series B22, 68-90 (1977).

17. Rose, D.J.
Tarjan, R.E.
Leuker, G.S. "Algorithms Aspects of Vertex Elimination of Directed Graphs". SIAM J. Comput. 5, 266-283 (1976).

18. Gavril, F. "Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independant Set of a Chordal Graph". SIAM J. Comput. 1, 180-187 (1972).

19. Morrow, C.
Goodman, S. "An Efficient Algorithm for Finding a Longest Cycle in a Tournament". Proc. 7th Southeastern Conference on Combinatorics, Graph Theory, and Computing, 453-462 (1976).

20. Rado, P.A. "A Note on Paths in Complete Directed Graphs". Bull. London Math. Soc., 2, 66-68 (1970).

21. Swamy, M.N.S.
Thalasiraman, K. "Graphs, Networks, and Algorithms". Chapter 5, John Wiley & Sons, New York, 1981.

22. Hopcroft, J.E.
Karp, R.M. "An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs". SIAM J. Comput. 2, 225-231 (1973).

23. Kuhn, H.W. "The Hungarian Method for the Assignment Problem". Naval Res. Logist. Quart. 2, 83-97 (1955).

24. Munkres, J. "Algorithm for the Assignment and Transportation Problems". J. SIAM, 5, 32-38 (1957).

25. Simon, F. "Sequencing and Scheduling". Chapter 10, John Wiley & Sons, New York, 1982.

26. Pnueli, A.
Lempel, A.
Even, S. "Transitive Orientation of Graphs and Identification of Permutation Graphs". Canad. J. Math. 23, 160-175.

27. Papadimitriou, C.H.
Steiglitz, K. "Some Complexity Results for the Travelling Salesman Problem". Proc. 8th Ann. ACM Symp. on Theory of Computing, Association for Computing Machinery, New York, 1-9, 1976.

28. Liu, C.L. "Introduction to Combinatorial Mathematics". McGraw-Hill, New York, 1968.

29. Harary, F. "Graph Theory". Addison-Wesley, Reading, MA, 1969.

30. Even, S.
Pnueli, A.
Lempel, A. "Permutation Graphs and Transitive Graphs". J. Assoc. Comput. Mach. 19, 400-410, 1972.

31. Gilmore, P.C.
Gomory, R.E. "Sequencing a One State-Variable Machine: A Solvable Case of the Travelling Salesman Problem". Operations Res. 12, 655-679 (1964).

32. Garfinkel, R.S. "Minimizing Wallpaper Waste, Part 1: A Class of Travelling Salesmen Problems". Operations Res. 25, 741-751, 1977.

33. Syslo, M.M. "A New Solvable Case of the Travelling Salesman Problem". Math. Programming 4, 347-348, 1973.

APPENDIX A

CONCEPTS OF GRAPH THEORY RELATIVE TO THE REPORT

1.0 SOME BASIC DEFINITIONS

1.1 Graphs

A Graph $G = (V,E)$ consists of two sets, a finite set V of elements called vertices and a finite set E of elements called edges. Each edge is identified with a pair of vertices.

Undirected graph. If the edges of G are identified with unordered pairs of vertices, then G is called an undirected graph or a nonoriented graph. An undirected graph is usually called a graph.

Directed graph. If the edges of G are identified with ordered pairs of vertices, then G is called a directed graph or an oriented graph, or a digraph. A digraph is usually denoted by $D = (V,A)$, and the directed edges are called arcs. A is the set of arcs.

All edges having the same pair of end vertices are called parallel edges. An edge having the same end vertices is called a self-loop. An undirected graph is called a simple graph if it has no parallel edges or self-loops. We only consider simple graphs in this report.

The graph that results after ignoring the orientations of a directed graph D is called the underlying undirected graph of D or the graph corresponding to the digraph D.

Let u, v be different vertices, the edge having the end vertices u, v is denoted uv (or vu) and the arc from u to v is denoted (u,v).

1.2 Other Definitions of Graph

In this report (section 2.4), another definition of a graph is involved.

A graph $G = (V,E)$ consists of vertex set V and edge set E. Each element of E is an ordered pair of vertices.

The following symbols are introduced:

$$E^{-1} = \{(u,v) \mid (v,u) \in E\},$$

$$\hat{E} = E \cup E^{-1}.$$

A graph $G = (V,E)$ is called an undirected graph if and only if $E = E^{-1}$ or equivalently, $E = \hat{E}$.

A graph D = (V,A) is called a directed graph if and only if $A \cap A^{-1} = \phi$. In addition, if $A + A^{-1} = E$, where plus sign expresses the union of two disjoint subsets, then D (or A) is called an orientation of G = (V,E).

1.3 Degrees

An edge is said to be incident on its end vertices. Two vertices are adjacent if they are the end vertices of the same edge. If two edges have a common end vertex, then the edges are said to be adjacent.

The number of edges incident on a vertex v is called the degree of the vertex, and it is denoted by d(v), or d_v.

A vertex of degree 1 is called a pendant vertex.

For a directed graph, an edge is said to be incident from its initial vertex and incident to its terminal vertex. The in-degree $d^-(v)$ or ${d^-}_v$ is the number of edges incident to v_j and the out-degree $d^+(v)$ is the number of edges incident out of v. The degree of v is $d_v = d_v^- + d_v^+$.

1.4 Weighted Graphs and Labelled Graphs

We can assign a weight w_{ij} to each edge v_iv_j (or arc (v_i,v_j)). A graph with weights associated with its edges is called a weighted graph.

On the vertex set V of a graph, a real-valued function f(v) can be defined. f(v) is called the label of the vertex v. A graph with labels associated with its vertices is called a labelled graph.

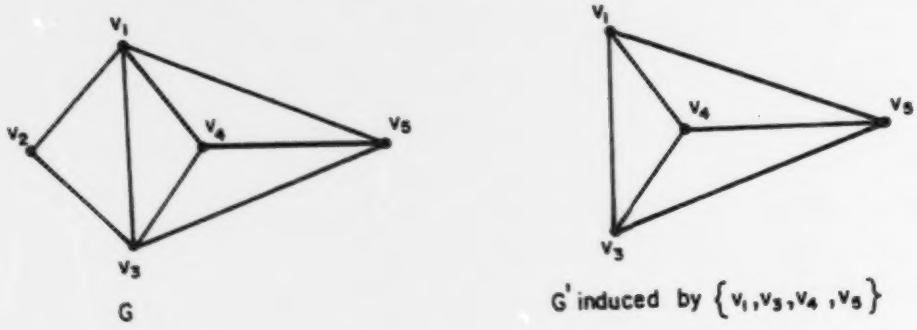
A weighted graph can be a labelled graph simultaneously. Some relation between the weights and labels can be established. For example,

$$f(v_i) + f(v_j) > w_{ij} \quad \forall \; v_i, v_j \in V .$$

1.5 Subgraphs

Consider a graph G = (V,E), G' = (V',E') is a subgraph of G if V' and E' are, respectively, subsets of V and E such that edge v_iv_j is in E' only if v_i, v_j are in V'. G' will be called a proper subgraph of G if either E' is a proper subset of E or V' is a proper subset of V. If all vertices of G are present in a subgraph G', then G' is called a spanning subgraph of G.

Let V' be a subset of vertex set V of a graph G = (V,E). The subgraph G' = (V',E) is the induced subgraph of G on the vertex set V' (or simply vertex-induced subgraph of G), if E' is a subset of E such that the edge v_iv_j is in E' if and only if v_i, v_j are in V'. In other words, if v_i, v_j are in V', then every edge having v_i and v_j as its end vertices should be in E'. Note that, in this case, V' completely specifies E' and thus the subgraph. The vertex-induced subgraph is called simply induced subgraph in this report. Figure A.1 shows a graph G and its induced subgraph G'.



G

G' induced by $\{v_1, v_3, v_4, v_5\}$

Fig. A.1

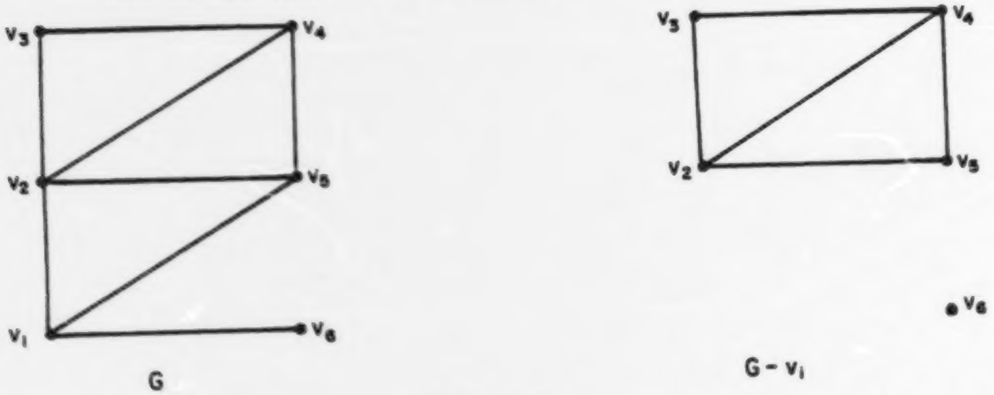
1.6 Operations on Graphs

We only consider the following operations on a graph.

Vertex removal. If v_i is a vertex of a graph, the G - v_i is the induced subgraph of G on the vertex set V - v_i; that is, G - v_i is the graph obtained after removing from G the vertex v_i and all the edges incident on v_i.

The removal of a set of vertices from a graph is defined as the removal of single vertices in succession.

The vertex removal is illustrated in Fig. A.2.



G

G - v_1

Fig. A.2

1.7 Representation of Graphs

Some matrices are associated with a graph and can be used to represent the graph. As examples, we introduce the incidence matrix for undirected graphs and the adjacency matrix for directed graphs.

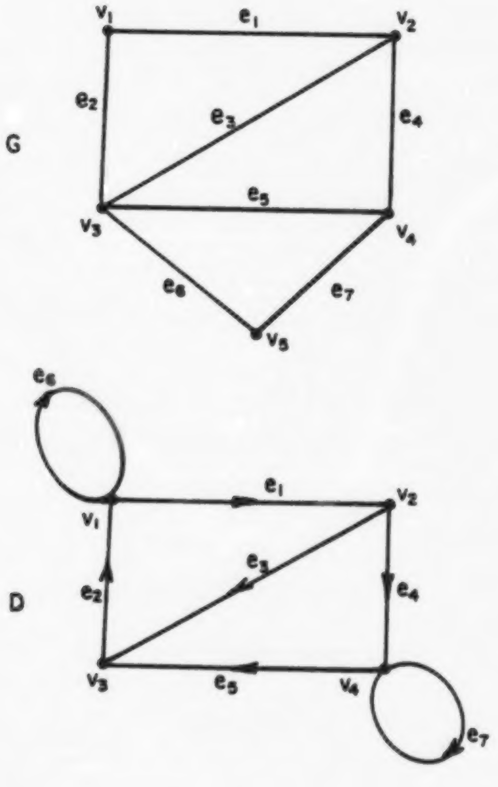
The incidence matrix $A = (a_{ij})$ of an undirected graph G is defined as follows:

$$a_{ij} = \begin{cases} 1 & \text{if the jth edge is incident on the ith vertex;} \\ 0 & \text{otherwise} \end{cases}$$

The adjacency matrix $M = (m_{ij})$ of a directed graph D is defined as follows:

$$m_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in A \text{ (A is the arc set);} \\ 0 & \text{otherwise} \end{cases}$$

Obviously, A is a $|V| x |E|$ matrix, M being $|V| x |V|$. Examples



$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

$$M = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

Fig. A.3

A graph can be represented in list fashion, say adjacency lists. For digraph D in Fig. A.3, the adjacency lists are shown in Fig. A.4.

	1	2	3	4
Head	1	3	4	6
Degree	2	1	2	2

	1	2	3	4	5	6	7
DATA	1	3	1	2	4	2	4

Fig. A.4

These lists show that, for example, the arcs incident on v_3 start with the 4th entry of list DATA and there are two such arcs: (v_3,v_2) and (v_3,v_4).

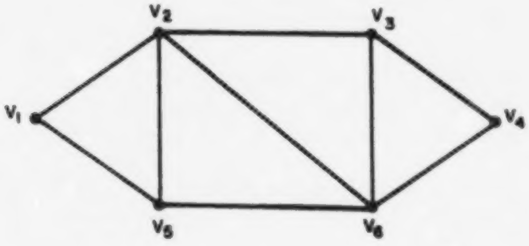
2.0 PATHS AND CIRCUITS

2.1 Paths

A walk in a graph $G = (V,E)$ is a finite alternating sequence of vertices and edges, v_0, e_1, v_1, $e_2,\dots,v_{K-1}$, e_K, v_K beginning and ending with vertices such that v_{i-1} and v_i are the end vertices of the edge e_i, $1 \leq i \leq K$. Alternately, a walk can be considered as a finite sequence of vertices $v_0,v_1,\dots,v_K$, such that $v_{i-1}\ v_i$, $1 \leq i \leq K$, is an edge in the graph G. Note that in a walk, edges and vertices can appear more than once.

A walk is open if its end vertices are distinct. Otherwise, it is closed. A walk is a trail if its edges are distinct. A trail is open if its end vertices are distinct. Otherwise, it is closed.

An open trail is a path if all its vertices are distinct.



a walk: $v_1v_2v_3v_2v_5v_6$
a trail: $v_1v_2v_3v_6v_2v_5$
a path: $v_1v_2v_3v_6v_5$

Fig. A.5

The number of edges in a path is called the length of the path.

If p is a path in graph G and there is no path in G with the length larger than that of p, then p is called the longest path in G. Obviously, the length of the longest path cannot exceed $|V|-1$.

A directed walk in a directed graph $D = (V,A)$ is a finite sequence of vertices $v_0, v_1, \dots, v_K$, such that (v_{i-1}, v_i), $1 < i < K$, is an arc in D. Thus, we can define directed path as defined as undirected graph: An open directed trail is a directed path if all its vertices are distinct. Note that, in Fig. A.6, $v_1v_5v_2v_3$ is directed path, but $v_4v_1v_5v_2$ is not.

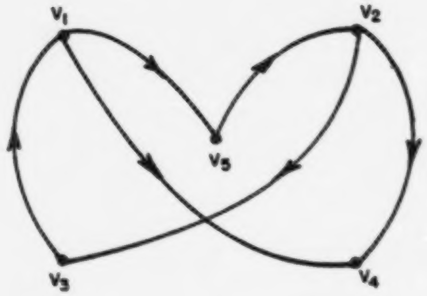


Fig. A.6

2.2 Circuits

A closed trail in a graph G is a circuit if all its vertices except the end vertices are distinct.

A closed directed trail in a digraph D is a directed circuit if all its vertices are distinct.

If C is a circuit in graph G, an edge in G joining two nonconsecutive vertices of the circuit is called a chord of C. If C has no chord, C is said to be nonchordal circuit or chordless circuit.

A chord can be defined for a closed walk or trail (or for directed case) as defined for circuits.

A circuit (or walk, trail) is even if it has odd vertices. Otherwise, it is odd. For example, $v_3v_1v_5v_2v_3$ in the digraph in Fig. A.6 is an even circuit. $v_1v_2v_3v_2v_5v_1$ in the undirected graph in Fig. A.5 is an odd walk.

2.3 Hamiltonian Paths and Hamiltonian Circuits

A Hamiltonian path in a graph G is a path in G containing all the vertices of G. A Hamiltonian circuit in G is a circuit containing all the vertices of G.

A graph is defined to be Hamiltonian if it has a Hamiltonian circuit. The graph G, shown in Fig. A.7 is Hamiltonian because the sequence of edges $e_1e_2e_3e_4e_5e_6$ constitutes a Hamiltonian circuit in G_1. The graph G_2 is not Hamiltonian. G_2 has a Hamiltonian path formed by the edges $e_1e_2e_3e_4$, but it has no Hamiltonian circuit.

If G is a Hamiltonian graph, it may have more than one Hamiltonian circuits.

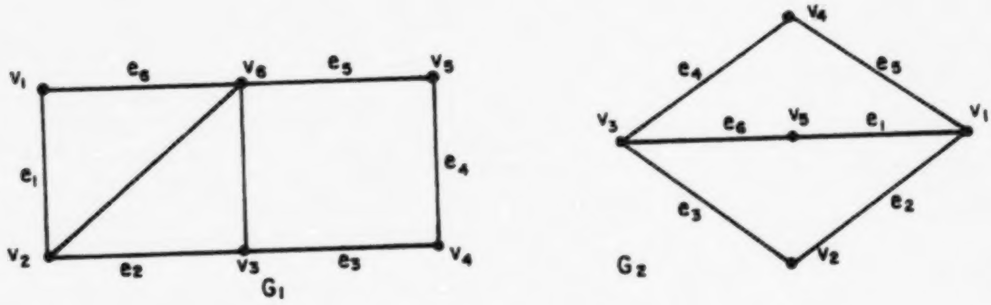


Fig. A.7

A directed circuit (path) in a directed graph D is a directed Hamiltonian circuit (path) of D if it contains all the vertices of D.

A directed graph is a directed Hamiltonian graph if it has a directed Hamiltonian circuit.

2.4 Vertex-Disjoint Paths

Two paths without a common vertex are called vertex-disjoint paths.

There are no vertex-disjoint paths of length more than 1 in G_2 of Fig. A.7. In G_1, $v_1v_2v_6$ and $v_3v_4v_5$, $v_2v_1v_6$ and $v_5v_4v_3$ are vertex-disjoint paths.

3.0 SPECIAL GRAPHS

3.1 Complete Graphs

A complete graph is a simple graph in which every pair of vertices is adjacent. It may be seen that a complete graph with n vertices has n(n-1)/2 edges. A directed graph D is complete if its underlying undirected graph is complete. A tournament is a complete directed graph.

A tournament with 5 vertices is shown in Fig. A.8.

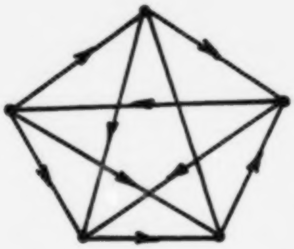


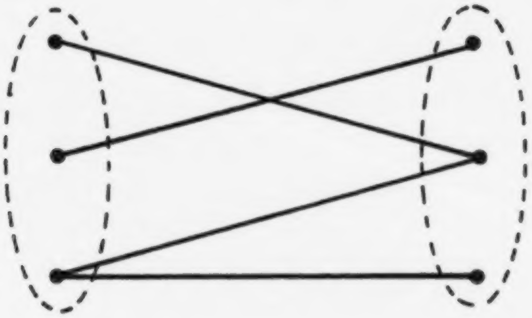
Fig. A.8

3.2 Bipartite Graph

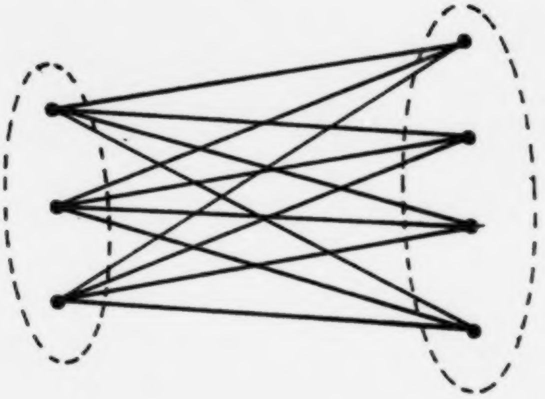
A graph G = (V,E) is bipartite if its vertex V can be partitioned into two subsets X and Y such that each edge of E has one end vertex in X and another in Y. (X,Y) is referred to as a bipartition, and G is usually denoted by G(X,Y) or G = (X,Y).

If in a simple bipartite graph G, with bipartition (X,Y), there is an edge $v_i v_j$ for every vertex v_i in X and every vertex v_j in Y, then G is called a complete bipartite graph.

A bipartite graph and a complete bipartite graph are shown in Fig. A.9. Sometimes, a bipartite graph is called simply bigraph.



a bipartite graph



a complete bipartite graph

Fig. A.9

3.3 Trees and Directed Tree

A graph is said to be acyclic if it has no circuits. A tree is a connected acyclic graph. The vertices having degree 1 are called pendant vertices. There exist several other equivalent ways to characterizing a tree. These are listed in the following theorem (the proof is omitted).

Theorem A.1 The following statements are equivalent for a graph G with n vertices and m edges:

1. G is a tree;
2. There exist exactly one path between any two vertices of G;
3. G is connected and $m = n-1$;
4. G is acyclic and $m = n-1$;
5. G is acyclic and any two nonadjacent vertices are connected by an edge then the resulting graph has exactly one circuit.

A vertex v of a directed graph D is a root of D if there is directed path from v to all the remaining vertices of D. A directed graph D is a tree if the underlying undirected graph is a tree. A directed graph D is a directed tree if D is a tree and has a root.

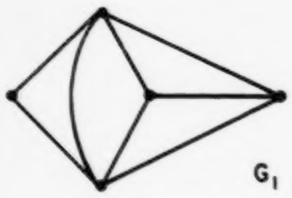
3.4 Comparability Graph and Chordal Graph

An undirected graph G possessing the following property is called a comparability graph.

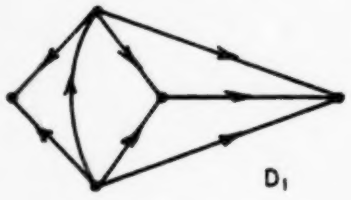
Transitive Orientation Property. Each edge be assigned a one-way direction in such a way that the resulting oriented graph $D = (V,A)$ satisfies the following condition:

$(u,v) \in A$ and $(v,w) \in A$ imply $(u,w) \in A$ $(\forall u,v,w \in V)$.

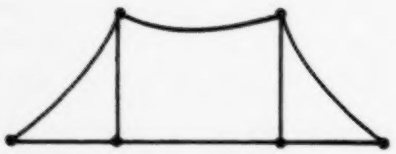
An undirected graph G is called chordal or triangulated if any circuit of length strictly greater than 3 possesses a chord, that is, an edge joining two nonconsecutive vertices of the circuit. Figure A.10 shows some illustrations of comparability graph and chordal graph.



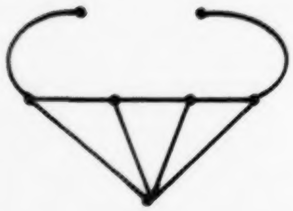
a chordal graph which is also a comparability graph



the transitive orientation of G_1



a comparability graph which is not a chordal graph



a chordal graph which is not a comparability graph

Fig. A.10

4.0 INDEPENDENT SET, CLIQUE AND MATCHING

4.1 Independent Set, Maximum Independent Set and Independent number

Assume that $G = (V,E)$ is an undirected graph. A subset S of V is called an independent set of G if no two vertices of S are adjacent in G. An independent set is also called a stable set.

An independent set S is maximal if there is no independent set which properly contains S as a subset. An independent set S of G is maximum if G has no independent set S' with $|S'|>|S|$. The number of vertices in a maximum independent set of G is called the independence number (stable number) of G.

For example, in the graph of Fig. A.11, the sets $\{b,d\}$, $\{b,f\}$, $\{a,c\}$ and $\{b,d,f\}$ are independent sets. Of these, $\{b,d\}$ and $\{b,f\}$ are not maximal independent sets; $\{a,c\}$ is maximal but it is not maximum, $\{b,d,f\}$ is maximum.

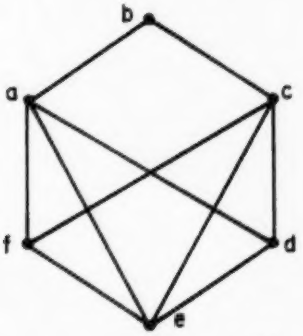


Fig. A.11

4.2 Clique, Minimum Clique Cover and Clique Cover Number

A clique of a graph $G = (V,E)$ is a set $C \subseteq V$ such that the induced subraph of G on C is a complete graph.

A clique cover of size K is a partition of the vertices $V = C_1 + C_2 + \ldots + C_K$ such that each C_i is a clique.

A clique C is maximal if there is no clique of G which properly contains C as a subset. A clique C is maximum if there is no clique of G of larger cardinality. The smallest possible clique cover of G is called the minimum clique cover.

The number of vertices in a maximum clique of G is called the clique number of G and the size of a minimum clique cover is called the clique cover number.

The following notations are used:

$\alpha(G)$: the independence number of G;
$w(G)$: the clique number of G;
$K(G)$: the clique cover number of G.

For example, in the graph of Fig. A.12, $\{a,b\}$, $\{a,b,e\}$, $\{a,b,e,d\}$, are clique, but $\{a,b\}$, $\{a,b,e\}$ are not maximal cliques. The maximum cliques are $\{a,b,e,d\}$ and $\{b,d,f,e\}$. $w(G) = 4$, $\{\{a,e\},\{b,d\},\{f,c\}\}$ is a clique cover, $\{\{a,b,e,d\},\{c,f\}\}$, $\{\{a,e,b\},\{e,f,d\}\}$ are all minimum clique cover. $K(G) = 2$.

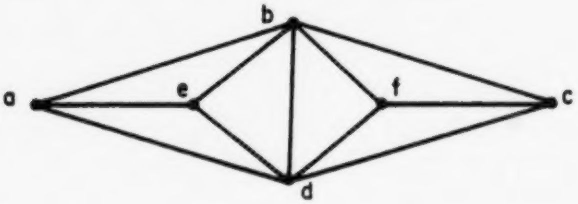
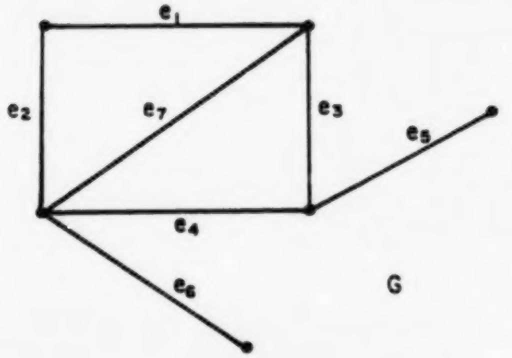


Fig. A.12

4.3 Matching, Maximum Matching and Matching Number

Two edges in graph G are said to be independent if they do not have any common vertex. A matching M in G is a set of independent edges. If there is no matching in G which properly contains M as a subset, M is a maximal matching of G. A matching with the largest number of edges is called a maximum matching and the size of the matching is called the matching number of G (denoted by M(G)).

Figure A.13 shows an example.



the minimum matching:
$\{e_1, e_5, e_6\}$
$M(G) = 3.$

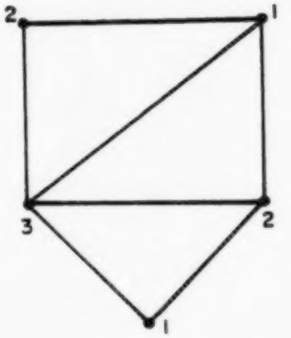
Fig. A.13

5.0 PERFECT GRAPH

5.1 Chromatic Number

A K-vertex coloring of a graph G is the assignment of K-distinct colors to the vertices of the graph. A vertex coloring is proper if in the coloring no two adjacent vertices receive the same color. A proper 3 vertex coloring is shown in Fig. A.14. A graph is K-vertex colorable if it has a proper K-vertex coloring.

The chromatic number $x(G)$ of a graph G is the minimum number K for which G is K-vertex colorable. G is K-chromatic if $x(G) = K$.



$x(G) = 3$

Fig. A.14

5.2 Perfect Graph

By the definition of clique and proper vertex coloring, it is easy to see that the vertices in a clique have the distinct colors in a proper vertex coloring. Therefore, for the general graph, we have

$$w(G) \leqslant x(G)$$

where, $w(G)$ and $x(G)$ are the clique number and chromatic number of G respectively.

Moreover, the intersection of a clique and an independent set of G can be at most one vertex. So, if an independent set of size α is covered by cliques, the size of the cliques cover is α at least. Therefore

$$\alpha(G) \leqslant K(G)$$

where $\alpha(G)$ and $K(G)$ are, respectively, the independence number and clique cover number of G.

Let G = (V,E) be an undirected graph. G is called a perfect graph if the following properties are satisfied:

$$w(G_A) = x(G_A) \quad (\forall A \subseteq V)$$

$$\alpha(G_A) = K(G_A) \quad (\forall A \subseteq V)$$

where G_A is the subgraph of G induced by A.

A much stronger result was conjectured by Berge, cultivated by Fulkerson and finally proven by Lovasy, namely, the two equations are equivalent (Perfect Graph Theorem).

APPENDIX B

SOME SPECIAL DEFINITIONS, THEOREMS, PROBLEMS AND ALGORITHMS IN GRAPH THEORY RELEATIVE TO THE REPORT

1.0 TRO Theorem

For a given undirected graph $G = (V,E)$, a polynomial time algorithm can be established for determining whether or not G is a comparability graph. This algorithm uses some concepts as a constructive tool, and the concepts are useful for section 3 of the report. They are introduced in [12] and some modifications, in symbol, are made here. The definition of graph in section 1.2 of Appendix A is used.

1.1 The Binary Relation Γ

Γ is a relation defined on E of an undirected graph $G = (V,E)$ (not necessarily a job compatibility graph), as follows:

$$(a,b)\ \Gamma\ (a',b') \quad \text{iff} \quad \begin{cases} \text{either } a = a' \text{ and } (b,b') \bar{\in} E \\ \text{or } \ b = b' \text{ and } (a,a') \bar{\in} E. \end{cases}$$

1.2 Implication Classes

The reflexive, transitive closure of Γ is easily shown to be an equivalence relation on E and hence partitions E into what we call the implication classes of G. Thus arcs (a,b) and (c,d) are in the same implication class if and only if there exists a sequence of arcs:

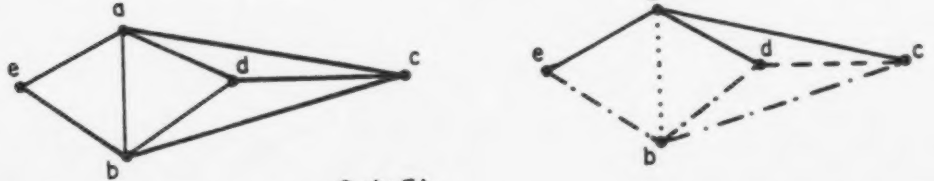
$$(a,b) = (a_0,b_0)\Gamma(a_1,b_1)\Gamma\ldots\Gamma\ (a_K,b_K) = (c,d) \text{ with } K > 0 \qquad \text{(B1)}$$

Such a sequence is called a Γ-chain.

For example, the graph G in Fig. B.1 has eight implication classes:

$$A_1 = \{(a,b)\},\ A_2 = \{(c,d)\},\ A_3 = \{(a,c),(a,d),(a,e)\},\ A_4 = \{(b,c),(b,d),(b,e)\}$$

$$A_1^{-1} = \{(b,a)\},\ A_2^{-1} = \{(d,c)\},\ A_3^{-1} = \{(c,a),(d,a),(e,a)\},\ A_4^{-1} = \{(c,b),(d,b),(e,b)\}$$



G =(V,E)

Fig. B.1

1.3 G-Decompositions and Decomposition Schemes

If A, B are disjoint sets (or subsets), we often write their union with a plus sign: A + B. Therefore, a partition of edge set E can be written: $E = E_1 + E_2 + \ldots + E_K$. We say that a partition of E is a G decomposition of E if B_i is an implication class of $E_i + E_{i+1} + \ldots + E_K$, for all $i = 1, \ldots K$. Where, each E_i is a subset of E, and B_i is an arc set of representatives from (a,b) and (b,a) for each edge ab belonging to E_i (i.e., $E_i = B_i \quad B_i^{-1}$). A sequence of arcs $\delta = ((x_1,y_1),\ldots(x_K,y_K))$ is called a decomposition if there exists a G-decomposition $E = E_1 + \ldots + E_K$ satisfying $(x_i,y_i) \in B_i$, for all $i = 1,\ldots K$.

We still consider the graph G in Fig. B.1. We have $E = \{(a,b),(b,a),(a,e),(e,a),(a,d),(d,a),(a,c),(c,a),(b,e),(e,b),(b,d),(d,b),(b,c),(c,b),(d,c),(c,d)\}$. Figure B.2 gives the G-decomposition and decomposition scheme. Where K = 3, $E_1 = \{(a,e),(e,a),(a,d),(d,a),(a,c),(c,a)\}$, $E_2 = \{(b,e),(e,b),(b,a),(a,b),(b,d),(d,b,)(b,c),(c,b)\}$, $E_3 = \{(d,c),(c,d)\}$ and $B_1 = \{(a,e),(a,d),(a,c)\}$, $B_2 = \{(b,e),(b,a),(b,d),(b,c)\}$, $B_3 = \{(d,c)\}$.

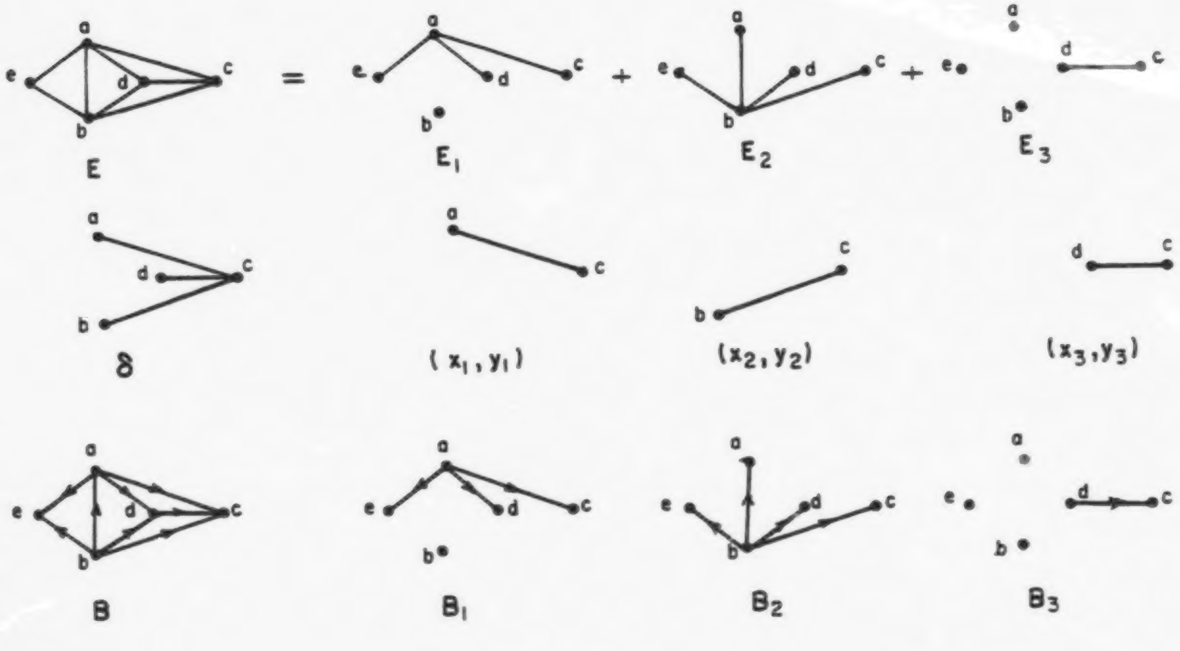


Fig. B.2

Contrasting Fig. B.1 and Fig. B.2, we have $B_1 = A_3$, $B_2 = A_4 + A_1^{-1}$, $B_3 = A_2^{-1}$.

The following theorem has been shown in [15], [16].

1.4 Theorem B.1 (TRO Theorem)

Let $G = (V,E)$ be an undirected graph with G-decomposition $E = E_1 + \ldots + E_K$. The following statements are equivalent:

(1) $A \quad A^{-1} = \emptyset$ for all implication class A of E;
(2) $B_i \quad B_i^{-1} = \emptyset$ for $i = 1,2,\ldots K$;
(3) Every closed walk of edges v_1v_2, $v_2v_3, \ldots v_gv_1 \ \varepsilon\ E$ such that $v_{g-1}v_1$, v_gv_2, $v_{i-1}v_{i+1}\ \bar{\varepsilon}\ E$ $(i=2,\ldots g-1)$ has even length;
(4) $G = (V,E)$ is a comparability graph.

Consider the graph in Fig. B.3, abcdcefeba is a closed walk with length 9 and without chords (ac,bd,de,cf,...$\bar{\varepsilon}$ E, E is the edge set), hence the graph is not a comparability graph.

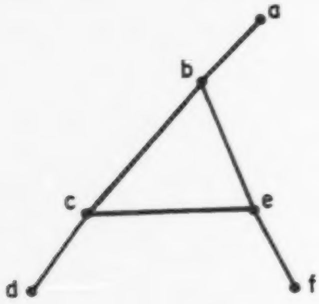


Fig. B.3

2.0 SOME CONCEPTS AND RESULTS ABOUT CHORDAL GRAPH

2.1 Simplicial Vertex

G is undirected graph. A vertex v of G is called simplicial, if its adjacency set $U(v)$ induces a complete subgraph of G, i.e. $U(v)$ is a clique.

Elimination Scheme. Let $\lambda = (v_1, \ldots v_n)$ be an order of the vertices of G. We say λ is a perfect vertex elimination scheme (simply, elimination scheme) if each v_i is a simplicial vertex of the induced subgraph $G_{\{v_i,\ldots v_n\}}$. In other words, each set

$$U'(v_i) = \{v_j \mid v_j \ \varepsilon\ U(v_i),\ j > i\} \tag{B.2}$$

is a clique.

For example, the graph G_1 in Fig. B.4 is a chordal graph, and a,d are the simplicial vertices, $\lambda = (abefcd)$ is an elimination scheme. G_2 is not a chordal graph (ebcd is a chordless circuit), and has a simplicial vertex a, also. G_3 is not a chordal graph and has no simplicial vertices.

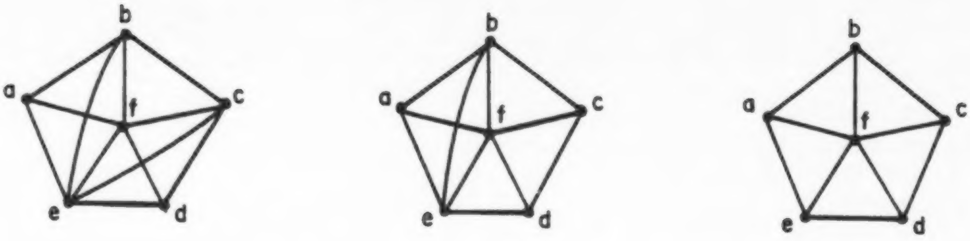


Fig. B.4

2.2 Vertex Separator

A subset S V is a vertex separator for nonadjacent vertices a and b if the removal of S from the graph separates a and b into distinct connected components. If no proper subset of S is an a-b separator, then S is a minimal vertex separator for a and b. Consider the graphs of Fig. B.4, in G_1, G_2, and G_3, $\{b,d,f\}$ is always a minimal separator for a,c.

Theorem B.2 Let G be an undirected graph. The following statements are equivalent.

1. G is chordal graph;
2. G has a perfect vertex elimination scheme. Moreover, any simplicial vertex can start a perfect scheme;
3. Every minimal vertex separator induces a complete subgraph of G.

2.3 Algorithm for Recognizing Chordal Graphs

A linear-time algorithm for recognizing chordal graphs has been given [17] as follows:

```
begin
assign the label $\phi$ to each vertex;
for $i \leftarrow n$ to 1 step -1 do;
select: pick an unnumbered vertex v with largest label;
$\sigma(i) \leftarrow v$; comment: this assigns to v the number i;
update: for each unnumbered vertex $w \in U(v)$ do add i to label (w);
end
```

The vertices are numbered from n to 1 in the order that they are selected in line 3. This numbering fixes the positions of an elimination scheme σ. For each vertex x, the label of x will consist of a set of numbers listed in decreasing order. The vertices can then be lexicographically ordered according to their label. Ties are broken arbitrarily.

Lexicographic order is simply dictionary order, so that $9761<985$ and $643<6432$, and so on.

The algorithm produces an elimination scheme σ. If σ is perfect, according to Theorem B.2, the given graph is a chordal graph.

2.4 Algorithm for Finding the Maximum Independent Set and Minimum Clique Cover [18]

Let σ be a perfect elimination scheme for $G = (V,E)$. We define a sequence of vertices $y_1, y_2, \ldots y_t$, inductively, in the following manner: $y_1 = \sigma(1)$; y_i is the first vertex in σ which follows y_{i-1} and which is not in $X_{y_1} \cup X_{y_2} \cup \ldots \cup X_{y_{i-1}}$, where X_{y_i} is the neighbour set of y_i. All vertices following y_t are in $X_{y_1} \cup \ldots \cup X_{yt}$. Hence $V = \{y_1, y_2, \ldots y_t\} \cup X_{y_1} \cup \ldots X_{yt}$.

The set $\{y_1, y_2, \ldots y_t\}$ is a maximum independent set since if edge $y_i y_j \in E$ for $i < j$, then $y_i \in X_{yj}$, and this is impossible. Thus $\alpha(G) > t$. On the other hand, each of the sets $y_i \cup X_{yi}$ is a clique, hence $\{y_1 \cup X_{y_1}, \ldots y_t \cup X_{yt}\}$ can give a clique cover. Thus $K(G) < t$. Since G is a chordal graph, we have $\alpha(G) = K(G) = t$. Therefore, we have produced a maximum independent set and a minimum clique cover.

3.0 COMPLEXITY OF SOME PROBLEMS

3.1 Hamiltonian Path and Circuit Problems

Assume that $G = (V,E)$ is an undirected graph.

Problem 1. Does G contain a Hamiltonian path?
Problem 2. Does G contain a Hamiltonian circuit?

Problem 1 and 2 are NP-complete for the general graph. Problem 2 remains NP-complete if a Hamiltonian path for G is given as part of the instance [27]. Solvable in polynomial time if the degree < 2 for each vertex [28].

Assume that $D = (V,A)$ is a directed graph.

Problem 3. Does D have a directed Hamiltonian path?
Problem 4. Does D have a directed Hamiltonian circuit?

Problem 3 and 4 are NP-complete. If D is a tournament, problem 3, 4 can be solved in polynomial time [19]. Problem 3 can be solved by polynomial time for acyclic digraphs [9].

3.2 Longest Path Problems

Assume that G is a weighted graph (undirected or directed), u,v are two specified vertices. Is there a path in G from u to v of length K (K is a positive integer) or more, i.e. whose edge weights sum to at least K?

The problem remains NP-complete if all the weights are 1. The general problem can be solved in polynomial time for acyclic digraphs [9].

3.3 Maximum Independent Set Problems

Assume that $G = (V,E)$ is an undirected graph and K is a positive integer. Does G contain an independent set of size K or more?

The problem is solvable in polynomial time for bipartite graph [29], for graphs with no vertex degree exceeding 2, for chordal graphs [18]. The problem of determining the independence number is polynomial solvable for comparability graphs [12].

3.4 Maximum Clique Problems

For an undirected graph $G = (V,E)$ and a positive integer K, is there a clique of size K and more? The problem is solvable in polynomial time for graphs for which a fixed degree bound is obeyed, for chordal graph [18], for comparability graphs [30].

3.5 Minimum Clique Cover Problems

Assume $G = (V,E)$ is an undirected graph and K is a positive integer. Does there exist a clique cover with size $k < K$? The problem remains NP-complete for all fixed $K > 3$. Solvable in polynomial time for $K < 2$, for chordal graph [18]. The clique cover number can be obtained in polynomial time for comparability graph [12].

3.6 Travelling Salesman Problems

In this report, the single crane scheduling problem involves the travelling salesman problem. The travelling salesman problem is NP-complete even if the distances $d(c_i,c_j) \in \{1,2\}$, for all cities $c_i,c_j \in C$. Some special cases that can be solved in polynomial time are discussed in [31-33].

4.0 ALGORITHMS FOR MATCHINGS

4.1 Maximum Bipartite Matching Algorithm (Hopcroft and Karp [22]

PROC-HOPKARP

H1. G is the given connected bipartite graph. Let M_0 be the null matching, that is, $M_0 = \emptyset$. Set $i = 0$.

H2. Construct from G the directed graph $\bar{G}$ by directing each unmatched edge (relative to M_i) of G from a girl to a boy, and each matched edge (relative to M_i) from a boy to a girl.

H3. Let L_0 be the set of free boys relative to M_i. If L_0 is not empty, go to step H4. Otherwise HALT. (The matching M_i is maximum).

H4. Construct as described below the sequence $L_0, L_1, \ldots$ of subsets of the vertex set V of $\bar{G}$ and the sequence $E_0, \bar{E}_1, \ldots$ of subsets of the edge set E of $\bar{G}$.

1. $E_i = \{(u,v) \mid (u,v) \varepsilon \bar{E},\ v \varepsilon L_i,\ u \bar{\varepsilon} L_0 \cup L_1 \cup \ldots \cup L_i),\ i = 0,1,2\ldots$
2. $L_{i+1} = \{u \mid \text{for some } v_i\ (u,v) \varepsilon E_i),\ i = 0,1,2,\ldots$
3. $m^* = \min\{i\} L'_i \cap \{\text{free girls}\} \neq \emptyset\}$.

H5. If m* is defined, go to step H6; otherwise HALT. (There is no augmenting path relative to M_i and hence M_i is a maximum matching.)

H6. Construct the subgraph $G^* = (V^*,E^*)$ of $\bar{G}$, where

$$V^* = L_0 \cup L_1 \cup \ldots\ L_{m^*-1} \cup \{(L_{m^*} \cap \{\text{free girls}\})\}$$

and

$$E^* = E_0 \cup E_1 \cup \ldots \cup E_{m^*-2}\ \{(u,v) \mid v \varepsilon L_{m^*-1} \text{ and } u \varepsilon \{\text{free girls}\}\}.$$

H7. Carry out the procedure PROC-AUGMENT and then go to step H2.

PROC-AUGMENT

A1. Set Point = 0. (Clear the array PATH.) Select a free girl in G*, which is not yet labeled "examined", and go to step A2. If no such free girl exists, go to step A8.

A2. Label y "examined". Set POINT = POINT + 1 and PATH(POINT) = y. (y is now placed on top of the array PATH.)

A3. Select an edge (y,x) which is not yet labeled "examined" and go to step A5. If no such edge exists (we can get no augmenting path which uses vertex y), do the following:

1. If POINT = 1, then go to step A1.
2. If POINT > 1, then go to step A4.

A4. Set POINT = POINT - 2 and y = PATH(POINT). (Remove the top two elements of the array PATH.) Go to step A3.

A5. Label (y,x) "examined". If x is already labeled "examined" (x already occurs in some shortest augmenting path or there is no such path which uses x), then go to step A3. Otherwise label x "examined" and set POINT = POINT + 1 and PATH(POINT) = x. (x is now placed on top of the array PATH.)

A6. If x is free relative to M_i, then go to step A7. Otherwise get y = mate of x and go to step A2.

A7. An augmenting path P relative to M_i is now found. The vertices of P are PATH(1), PATH(2),...,PATH(POINT). Set $M_{i+1} = M_i \oplus P$. (M_{i+1} is the new matching after augmentation.) Set i = i + 1 and go to step A1.

A8. (All shortest vertex-disjoint augmenting paths in G* have been found and the corresponding augmentations done.) End the procedure.

4.2 Optimal Assignment Algorithm (Kuhn and Munkres [23,24]

S1. G is the given complete bipartite graph with bipartition (X,Y) and $|X| = |Y|$. $W = [w_{ij}]$ is the given weight matrix. Set i = 0.

S2. Start with an arbitrary feasible vertex, labeling f in G. Find the equality subgraph G_j and then select an initial matching M_i in G_j.

S3. If all the vertices in X are saturated in M_i, then M_i is a perfect matching, and hence is an optimal matching. So HALT. Otherwise let u be an unsaturated vertex in X. Set S = (u) and $T = \emptyset$.

S4. Let $\Gamma_f(S)$ be the set of vertices which are adjacent in G_f to the vertices in S. If $\Gamma_f(S) \supset T$, then go to step S5. Otherwise (that is, if $\Gamma_f(S) = T$) compute

$$d_f = \min_{\substack{x \varepsilon S \\ y\ T}} \{f(x) + f(y) - w(x,y)\}$$

and get a new feasible vertex labeling f' given by

$$f'(v) = \begin{cases} f(v) + d_f, & \text{if } v \varepsilon S \\ f(v) + d_f, & \text{if } v \varepsilon T. \\ f(v), & \text{otherwise} \end{cases}$$

(Note that $d_f > 0$ and $\Gamma_f(S) = T$.)
Replace f by f' and G_f by $G_{f'}$.

S5. Select a vertex y from $\Gamma_f(S) - T$. If y is not saturated in M_i, go to step S6. Otherwise set z = mate of y in $M_i S = S \ \{z\}$ and $T = T \cup \{y\}$, and then go to step S4.

S6. (An augmenting path P is found.) Set $M_{i+1} = M_i \oplus P$ and $i = i + 1$. Go to step S3.

REPORT DOCUMENTATION PAGE/PAGE DE DOCUMENTATION DE RAPPORT

REPORT/RAPPORT
1a TR-SYS-005

REPORT/RAPPORT
1b NRC NO. 23528

REPORT SECURITY CLASSIFICATION
CLASSIFICATION DE SÉCURITÉ DE RAPPORT
UNCLASSIFIED
2

DISTRIBUTION/DIFFUSION
UNLIMITED
3

TITLE/SUBTITLE/TITRE/SOUS-TITRE (CLASSIFICATION) Unclassified
Algorithms For Solving Crane Scheduling Problems
4

AUTHOR(S)/AUTEUR(S)
Jiesheng Wang
Visiting Scholar from
Institute of Computing Technology
Chinese Academy of Sciences
P.O.B 2704
Beiging, China
5

SERIES/SÉRIE
Technical Report
6

CORPORATE AUTHOR/PERFORMING AGENCY/AUTEUR D'ENTREPRISE/AGENCE D'EXÉCUTION
Systems Laboratory
Division of Mechanical Engineering
7

SPONSORING AGENCY/AGENCE DE SUBVENTION
8

DATE	FILE/DOSSIER	LAB. ORDER COMMANDE DU LAB.	PAGES	DIAGS	REFS
9 1984/07	10	11	12a 102	12b 69	12c 33

NOTES
13

DESCRIPTORS(KEY WORDS)/MOTS-CLÉS
Cranes, scheduling, algorithms, graph theory
14

SUMMARY/SOMMAIRE
Two approximate methods are presented for solving crane scheduling problems based on graph theory concepts.
15

ADDRESS/ADRESSE
P.A. Hamill, Head
Systems Laboratory
Division of Mechanical Engineering
National Research Council Canada
Montreal Road, Ottawa, Ontario K1A 0R6 (613) 993-9208
16